Uma Proposta de Framework para Sistemas de Controle para Plataformas de Robótica Social

Marcelo Rocha
MídiaCom Lab
Universidade Federal Fluminense
Niterói, Brazil
marcelo_rocha@midiacom.uff.br

Jesus Favela
Department of Computer Science
Centro de Investigación Científica y de Educación Superior de Ensenada
Ensenada, Mexico
favela@cicese.mx

Débora C. Muchaluat-Saade
MídiaCom Lab
Universidade Federal Fluminense
Niterói, Brazil
debora@midiacom.uff.br

ABSTRACT

There is a gap in open-source robotic platforms that can be customized according to application needs and available resources. Software systems for robots are usually complex due to the need to control multiple sensors and actuators in real time, while simultaneously and asynchronously performing tasks and responding to unexpected situations. In this context, a well-designed architecture that could facilitate building and using robots is desirable. This work proposes a framework for the development of control systems for social robotics platforms. The framework is modular and easily extensible, as it uses an object-oriented approach and adopts the publish-subscribe paradigm for asynchronous message exchange between modules. The framework proposed in this work was instantiated for the design and implementation of two open-source social robots, demonstrating its adaptability and extensibility in different scenarios. We expect that this work will contribute to popularizing the use of low-cost social robots in the areas of education and healthcare.

KEYWORDS

Robôs sociais, Framework para controle de robôs, Arquitetura para robôs sociais, EVA, FRED

1 INTRODUÇÃO

A adoção de novas tecnologias na educação pode estimular o engajamento de crianças e jovens, enquanto fazê-lo nos cuidados de saúde pode ajudar a conter custos e aumentar a qualidade de vida da população. Enquanto a robótica tradicional já é utilizada há bastante tempo nas escolas e no auxílio a procedimentos médicos como cirurgias, os Robôs Sociais [11], que enfatizam a interação social com os usuários como sua principal funcionalidade, têm sido cada vez mais desenvolvidos e adotados. A utilização desses dispositivos pode promover vocações em STEAM (*Science, Technology, Engineering, Arts, Math*), pois exigem habilidades em *hardware* e *software*, além de criatividade para programar o comportamento do robô. Da mesma forma, a interação social desempenha um papel importante na saúde, seja na orientação de intervenções terapêuticas ou no monitoramento das condições de saúde para diagnóstico precoce e avaliação do tratamento. No entanto, uma questão importante que impede que intervenções utilizando robôs sociais sejam amplamente concebidas e adotadas é o custo financeiro. Esses dispositivos são caros e dificilmente são utilizados em escolas ou ambientes de saúde, especialmente em países em desenvolvimento. Assim, existe uma lacuna nas plataformas robóticas de código aberto que podem ser personalizadas de acordo com as necessidades e recursos disponíveis para o desenvolvedor e usuários-alvo.

Os sistemas de *software* para robôs são normalmente complexos. Esta complexidade surge em grande parte da necessidade de controlar vários sensores e atuadores em tempo real. Os sistemas robóticos devem executar tarefas enquanto monitoram e respondem a situações inesperadas. Fazer tudo isso de forma simultânea e assíncrona aumenta muito a complexidade do sistema. A implementação de uma arquitetura bem projetada, juntamente com ferramentas de programação que suportem essa arquitetura, muitas vezes pode ajudar a gerenciar a complexidade. Atualmente, não existe uma arquitetura única que seja ideal para todas as aplicações, cada uma tem suas próprias vantagens e desvantagens [20]. Compreender esses pontos fortes e fracos é crucial ao selecionar uma abordagem arquitetônica para uma aplicação específica.

Este trabalho apresenta uma proposta de um *framework* para desenvolvimento de sistemas de controle para plataformas de robótica social. O *framework* apresenta um estrutura modular e de fácil extensão possuindo como única dependência a instalação de um *broker* MQTT [12]. Ele adota uma abordagem orientada a objetos e utiliza o paradigma *publish-subscribe* de troca de mensagens assíncronas entre os seus módulos.

O *framework* proposto neste trabalho foi instanciado para o projeto e implementação de dois robôs sociais de código aberto, mostrando suas capacidades de adaptação e extensão a diferentes cenários. Espera-se que este trabalho contribua para popularizar o uso de robôs sociais de baixo custo nas áreas de educação e saúde.

O restante do texto está organizado da seguinte forma. A Seção 2 discute os trabalhos relacionados. A Seção 3 apresenta o *framework* posposto. A Seção 4 demonstra o processo de instanciação do *framework* aplicado a dois cenários concretos envolvendo as plataformas de robótica social *open-source* EVA e FRED. A Seção 5 conclui este artigo e apresenta os trabalhos futuros.

2 TRABALHOS RELACIONADOS

De certa forma, pode-se considerar as arquiteturas de robôs como engenharia de *software* [20]. No entanto, as arquiteturas de robôs se distinguem de outras arquiteturas de *software* devido às necessidades especiais dos sistemas robóticos. Esses sistemas devem

In: Proceedings of the Brazilian Symposium on Multimedia and the Web (WebMedia'2024). Juiz de Fora, Brazil. Porto Alegre: Brazilian Computer Society, 2024.

ISSN 2966-2753

interagir de forma assíncrona e em tempo real com ambientes incertos e dinâmicos. A escolha do *framework* adequado para auxiliar no desenvolvimento de sistemas para robôs depende dos requisitos específicos do projeto como a complexidade dos comportamentos robóticos desejados e a integração de sensores e atuadores.

O projeto do *framework* OROCOS [3] busca desenvolver um *software* mais aberto, que seja adaptável e que possa atender a todas as necessidades intrínsecas ao desenvolvimento de *software* de controle para robôs. Além disso, ele busca atender às necessidades de quatro categorias de usuários: usuários finais, desenvolvedores de aplicações, desenvolvedores de componentes e desenvolvedores de *frameworks*. Dao [8] propôs um *framework* denominado OROMACS para a construção de um sistema de controle multiagente hierarquicamente estruturado que tem como base o OROCOS.

O ROS (*Robot Operating System*) [17] não é um sistema operacional no sentido tradicional de gerenciamento e agendamento de processos. Segundo Martin et al. [14], o ROS tornou-se, de fato, o *framework* padrão para o desenvolvimento de *software* em robótica. Ele é uma estrutura flexível de código aberto para a escrita de *software* para robôs que fornece uma coleção de mecanismos de comunicação, ferramentas, bibliotecas e regras que visam simplificar a tarefa de criação de *software* para várias plataformas robóticas [1].

Szücs et al. [22] propuseram um *software* de treinamento baseado em um robô humanoide amplamente utilizado na literatura, o robô NAO, da empresa Aldebaran Robotics. O *software* foi construído utilizando-se o *framework* NAOqi. O *framework* busca atender às necessidades de programação como paralelismo, sincronização, gerenciamento de eventos e gerenciamento de recursos. Além disso a Choregraphe Suite [21], uma IDE fornecida pela empresa SoftBank Robotics para programação dos robôs NAO e Pepper, se integra perfeitamente ao *framework* NAOqi.

O *framework* YARP (*Yet Another Robot Platform*) [15] foi escrito por e para pesquisadores em robótica humanoide que se deparam com uma pilha complicada de *hardware* para controlar e com uma pilha igualmente complicada de *software*. Ele oferece suporte a múltiplos protocolos de comunicação e flexibilidade para integrar componentes heterogêneos.

Ritschel [19] e Ritschel et al. [18] apresentam uma abordagem para implementar os movimentos do robô Reeti de maneira paralela e independente. A proposta utiliza o *framework* URBI (*Universal Robot Body Interface*) [2] que é baseado em uma arquitetura cliente/servidor de maneira que o servidor roda no robô e é acessado, normalmente, via TCP/IP. O *framework* inclui uma linguagem de *script* utilizada pelo cliente que é capaz de controlar as articulações do robô ou acessar seus sensores, câmera, alto-falantes etc.

Os trabalhos supracitados trazem contribuições importantes para a robótica, porém, apresentam limitações. Apesar da abrangência e do poder do ROS [17] sua curva de aprendizagem pode ser íngreme para iniciantes. Canela et al. [4] conduziram um estudo abordando os desafios encontrados no aprendizado do ROS. O estudo apresenta, na forma de um mapa mental, vários problemas relacionados a diversos fatores, entre eles: problemas de concorrência (problemas de memória compartilhada em funções *callbacks* e perda de mensagens), problemas gerados pelo impacto na frequência no envio de mensagens (*publish-subscribe*) e a complexidade na definição de novos formatos de mensagem pois esse processo requer a alteração de código em vários lugares em arquivos diferentes. Além disso, a sua gestão de dependências entre pacotes pode ser complicada e a compatibilidade entre versões pode causar problemas. O *framework* OROCOS [3] é complexo de instalar e configurar devido suas múltiplas dependências e requisitos de sistema. Ele exige do usuário conhecimento aprofundado de controle em tempo real e programação de componentes. Os *frameworks* NAOqi [22] e a IDE Choregraphe Suite [21] foram projetados exclusivamente para os robôs NAO e Pepper sendo ferramentas de *software* proprietário e de custo elevado. O *framework* YARP [15] apresenta um certo grau de complexidade em sua configuração inicial e na integração de diferentes módulos. A sua flexibilidade, dando suporte a múltiplos protocolos de comunicação, pode causar *overhead* gerando um impacto negativo em aplicações de tempo real. Por possuir uma linguagem de *script* própria, a utilização do URBI [2] exige que o usuário aprenda *urbiscript* e isso pode ser uma barreira para usuários familiarizados com outras linguagens.

Este trabalho apresenta uma proposta de *framework* para desenvolvimento de sistemas de *software* de controle para plataformas de robótica social. O *framework* é de fácil instalação e configuração com apenas uma dependência, a instalação e configuração de um *broker* MQTT. Isso reduz significativamente o tempo e o esforço necessários para que um usuário iniciante na área de desenvolvimento de *software* para robôs possa começar a desenvolver seu sistema de controle. Além disso a utilização de uma arquitetura orientada a objetos combinada com a comunicação baseada em MQTT facilita a integração de novos módulos e dispositivos permitindo uma adaptação rápida às mudanças e a novos requisitos de projeto.

3 FRAMEWORK PROPOSTO

Segundo Pressman e Maxim [16], um *Framework* é uma "miniarquitetura" reutilizável que serve como base para e a partir da qual outros padrões de projeto podem ser aplicados. Ele pode ser visto como uma estrutura (um arquétipo) de um sistema que é instanciado e especializado para gerar uma família de aplicações [10], sendo empregado para representar um problema de domínio específico, podendo servir como uma solução potencial para o problema em questão. Neste trabalho, foi adotada uma abordagem de especificação orientada a objetos para facilitar o entendimento da estruturação do *framework*. Essa estrutura é genérica e deve ser especializada para cada cenário específico. A Figura 1 apresenta os componentes do *framework* com suas classes e interfaces.

As interfaces apresentadas na Figura 1 definem algumas operações que devem ser implementadas pelas classes (módulos de controle) do sistema do robô. A interface **IRobotModule**, por exemplo, é responsável por definir comportamentos e métodos que são comuns a quaisquer módulos do sistema robótico que se deseja desenvolver, sejam eles, um módulo de controle de algum *hardware*, uma funcionalidade específica do robô ou um módulo de controle geral do sistema. Há também duas interfaces (**IBlocking** e **IUnBlocking**) que modelam um comportamento de bloqueio e desbloqueio para a implementação de módulos que executam suas tarefas de maneira síncrona com outros módulos. A implementação ou não dessas interfaces estará diretamente ligada a função

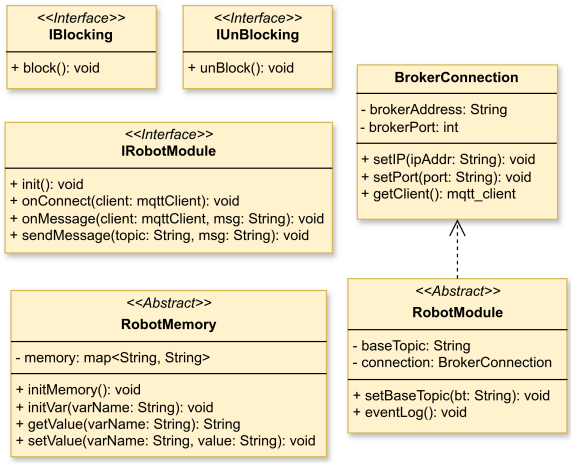

Figura 1: Componentes do *framework* proposto.

específica do módulo do robô sendo desenvolvido. Esse comportamento será exemplificado e detalhado através de um diagrama de sequência na Seção 4.2.

Além do arcabouço básico para a criação de módulos de controle para cada funcionalidade específica de um robô como o controle de atuadores, a leitura de sensores ou a requisição de serviços (locais ou na nuvem), o *framework* provê uma base para a criação de tipos de módulos de controle mais gerais. Um exemplo desse tipo seria um módulo *Script Player* que pode ser utilizado para executar comandos provenientes de alguma linguagem de *script*. Além disso, o *framework* disponibiliza uma classe abstrata **RobotMemory** que modela um sistema básico de memória definindo comportamentos intrínsecos a esse tipo de elemento como inicialização, criação, definição e alteração de valores de variáveis na memória do robô.

O estilo arquitetural adotado nesta proposta é baseado no paradigma *publish-subscribe*. A utilização desse padrão permite que os módulos executem e se comuniquem de maneira assíncrona e distribuída. Sendo assim, cada módulo instanciado a partir do *framework* deve possuir uma instância de um cliente MQTT a fim de que seja possível a comunicação entre os módulos e o *broker* MQTT. Um cliente MQTT se torna disponível a um módulo a partir do momento que esse módulo implementa a classe abstrata **RobotModule** que contém uma instância da classe concreta **BrokerConnection**.

A próxima seção descreve com detalhes a instanciação do *framework* proposto em dois cenários concretos: os sistemas de controle dos robôs EVA e FRED.

4 CENÁRIOS DE INSTANCIAÇÃO DO FRAMEWORK

Nesta seção, dois cenários de utilização do framework proposto são apresentados, a fim de exemplificar o seu uso. O primeiro cenário demonstra a instanciação das classes básicas utilizadas pata a criação do sistema de controle do robô social EVA, com todas a suas funcionalidades, e um *software* capaz de executar *scripts* na sua linguagem de programação EvaML [7]. O segundo cenário usa como base a plataforma de robótica *open-source* FRED. O cenário serve para exemplificar como a mesma estrutura utilizada para o

Figura 2: Plataforma EVA de robótica social de código aberto.

EVA pode ser utilizada para a construção de um sistema de controle para uma outra plataforma que possui uma arquitetura básica de *hardware* bem mais simples que a do robô EVA.

4.1 Funcionalidades do Robô EVA

O EVA, que pode ser visto na Figura 2, tem sido usado para orientar intervenções terapêuticas para pessoas com demência e auxiliar os cuidadores na abordagem de comportamentos perturbadores. O robô EVA é uma plataforma de robótica social de *open-source* projetada para apoiar pesquisas na área de interação Humano-Robô [6]. Ele possui diversas funcionalidades e elementos de comunicação verbal e não verbal [5]. Além disso, o EVA possui uma linguagem de programação declarativa baseada em XML, chamada EvaML [7], e um *software* de controle, o EvaSIM, [13] que pode executar *scripts* EvaML controlando o robô físico. Cada uma dessas funcionalidades foi implementada utilizando o *framework* proposto neste trabalho. A seguir será apresentada uma breve descrição de suas funcionalidades.

Expressões do Olhar. O EVA pode expressar a seguintes emoções através do seu *display*: neutra, raiva, nojo, medo, felicidade, tristeza e surpresa.

Comunicação Verbal. O EVA pode falar utilizando recursos da nuvem como o serviço de *Text-To-Speech* (TTS) do *IBM Watson*. Ele também pode transformar voz em texto utilizando o serviço na nuvem da API de *Speech-To-Text* (STT) do *Google*.

Movimentação da Cabeça e dos Braços. O EVA é capaz de mover a sua cabeça com dois graus de liberdade: para cima e para baixo, para esquerda e para a direita. Além da movimentação da cabeça, o EVA pode mover seus braços para cima e para baixo, podendo também, executar o movimento de *shaking* sacudindo os braços em torno de suas posições correntes.

Animações com os LEDs RGB. O EVA pode executar diversas animações com o conjunto de LEDs RGB em seu tórax. Essas animações utilizam várias cores pré-determinadas e tem como o objetivo, através de comunicação não verbal, expressar emoções.

Controle da Lâmpada Inteligente. O robô possui a capacidade de controlar efeitos sensoriais de luz através do controle uma lâmpada inteligente.

Player de Áudio. O EVA pode tocar vários tipos de arquivos de áudio como músicas, efeitos sonoros e os áudios gerados pelo

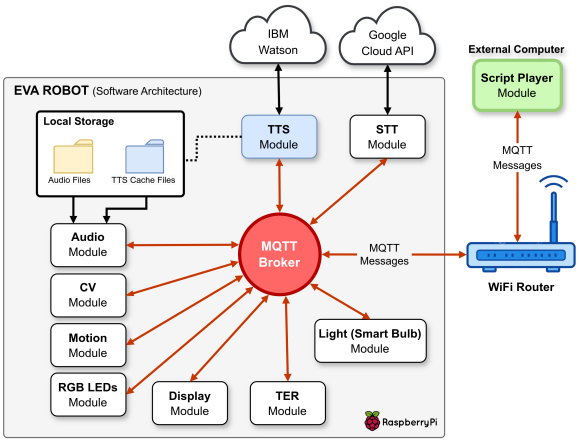

Figura 3: Arquitetura de *software* do robô EVA implementada a partir do *framework* proposto.

serviço de TTS do *IBM Watson*, para isso, ele utiliza uma caixa de som externa.

Visão Computacional. O EVA possui uma câmera acoplada à sua cabeça e a partir das imagens capturadas por ela, o sistema do robô implementa três funcionalidades: o reconhecimento da face do usuário, o reconhecimento de expressões faciais e a leitura de QR Codes.

4.2 Instanciação do *Software* do EVA

Como foi dito anteriormente, os módulos de controle definidos a partir do *framework* utilizam o paradigma *publish-subscribe* de troca de mensagens assíncronas. Cada módulo de controle do robô, que implementa cada uma de suas funcionalidades, é controlado através de mensagens que são definidas pelo desenvolvedor do sistema. Essas mensagens também são utilizadas como um mecanismo de sincronização entre os módulos, além de permitir o compartilhamento de informações relacionadas aos seus serviços. Os valores provenientes da leitura de sensores como um microfone (no caso específico de um módulo que transforma áudio em texto) ou de algum módulo capaz de classificar a emoção contida em um texto podem ser passados de um módulo para o outro através da troca de mensagens publicadas em tópicos específicos destinados a este fim.

O *software* de controle do EVA é composto por dez módulos. Nove deles, executam dentro de uma placa *Raspberry PI 4*, que fica embutida dentro do corpo impresso em 3D do robô. O décimo módulo roda em uma máquina separada. Os nove módulos são responsáveis por controlar cada uma das funcionalidades descritas na Seção 4.1. Todos os módulos foram implementados tendo como base o *framework* proposto neste trabalho. A Figura 3 permite que se tenha uma visão geral desses módulos e o esquema de comunicação de cada um deles com o *broker* MQTT.

Com base no comportamento de cada módulo do sistema do EVA, é possível dividi-los em três grupos distintos: I) os módulos que realizam tarefas de maneira assíncrona, sem fazer uso de algum mecanismo de sincronização com outros módulos, ou seja, eles recebem um comando de ativação e realizam suas tarefas sem sinalizar o seu término; II) os módulos que realizam tarefas síncronas e que retornam algum tipo de resposta (em formato de texto) sinalizando seu término através de uma mensagem de desbloqueio; III) os módulos que realizam tarefas de maneira síncrona, assim como os módulos do grupo II, porém, sem retornar qualquer tipo de resposta em formato textual. Como os processos de instanciação dos módulos pertencentes a cada grupo são similares entre si, esses processos serão descritos e discutidos a partir de cada um dos três grupos. A Tabela 1 apresenta cada grupo e seus módulos.

Tabela 1: Tópicos e mensagens para a implementação do sistema de controle do EVA.

Grupo	Módulo	Tópico (Assinado)	Mensagem
I	Display	evaEmotion	"HAPPY", "FEAR", "SAD" etc.
	Light	light	"BLACK|OFF", "BLUE |ON" etc.
	RGB LEDs	leds	"HAPPY", "ANGRY" etc.
	Motion	motion/head	"UP", "DOWN", "LEFT" etc.
		motion/arm/left	"POS0", "POS1", SHAKE1 etc
		motion/arm/right	"POS0", "POS1", SHAKE1 etc
II	STT	listen	Empty text
	TER	textEmotion	"Text"
	Computer Vision	userEmotion	Empty text
		userID	Empty text
		qrRead	Empty text
III	TTS	talk	"Voice|Text"
	Audio	sound	"file_name|BLOCKING"
		speech	"file_name"
	Script Player	"Ver Tabela 2"	"Ver Tabela 2"

4.2.1 ***Módulos - Grupo I.*** O primeiro grupo é formado pelos módulos *Motion*, responsável pelo controle da movimentação da cabeça e dos braços do robô, pelo módulo *RGB LEDs*, que executa animações com os LEDs RGB no tórax do EVA, pelo módulo *Light*, que controla a Lâmpada Inteligente, e pelo módulo *Display*, que mostra expressões do olhar do robô através de um *display* de 5.5". Todos os módulos desse grupo iniciam a execução de suas tarefas a partir do recebimento de um comando de ativação, proveniente de outro módulo. Os comandos de ativação são mensagens especificas publicadas no *broker* em tópicos determinados pelo desenvolvedor do sistema. Após sua execução, um módulo do grupo I não retorna qualquer informação como resposta e nem utiliza qualquer mecanismo que possa sinalizar o término de sua tarefa para o módulo que o controla. Um módulo de controle do robô como o módulo *Script Player*, que será visto mais adiante, pode enviar um comando para o módulo do *Display* fazendo com que o EVA mostre uma expressão de alegria. O módulo de controle atinge seu objetivo publicando a mensagem "HAPPY" no tópico *evaEmotion*, previamente assinado pelo módulo do *Display*. A Figura 4 apresenta todo o processo descrito junto com os elementos envolvidos.

A Figura 5 apresenta a instanciação do módulo *Display*. Basicamente, para implementar um módulo utilize o *framework* proposto, é necessário que o módulo seja desenvolvido seguindo os três passos:

(1) Estender a classe abstrata **RobotModule**. Ao fazer isso, o módulo passa a ter a capacidade de definir um tópico base que será usado como o tópico raiz do sistema do robô. Além

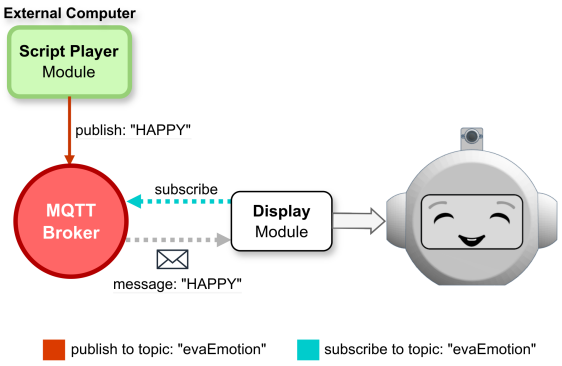

Figura 4: Controle das expressões do olhar do robô através da publicação da mensagem "HAPPY" no tópico *evaEmotion*.

disso, ele se torna capaz de instanciar um objeto de conexão com o *broker* MQTT.

(2) Instanciar o objeto de conexão com o *broker* obtendo um objeto do tipo *mqtt_client*. Isso permite que o módulo seja capaz de "assinar" e/ou "publicar" nos tópicos de seu interesse.

(3) Implementar a interface **IRobotModule**. No contrato da interface ela determina que todo módulo que pretende realizá-la deve implementar os métodos: *onConnect()*, que é o local onde o módulo deve assinar os tópicos de seu interesse; o método *onMessage()*, que é onde o módulo recebe e trata as mensagens publicadas nos tópicos por ele assinados; o método *sendMessage()*, que deve implementar o envio (a publicação) de mensagens, caso o módulo retorne algum valor e, por último, o método *init()*, que deve ser implementado a fim de instanciar o objeto de conexão com o *broker*, obtendo o cliente *mqtt* e anexando a ele as funções *callback* supracitadas: *onConnect()* e *onMessage*().

Os demais módulos do grupo I, por apresentarem o mesmo padrão de comportamento, seguem o mesmo processo de instanciação do módulo *Display*.

4.2.2 ***Módulos - Grupo II***. Os módulos que fazem parte deste grupo, são: o módulo *STT* (*Speech-To-Text*), que retorna um texto a partir do áudio da fala do usuário, o módulo *CV* (Visão Computacional), que executa três funcionalidades baseadas nas imagens capturadas através da *PiCamera*, retornando suas respostas em formato de texto, e o módulo TER (*Text-Emotion-Recognition*), que classifica um texto retornando a emoção contida nele. A resposta retornada pelo módulo *TER* também é em formato de texto. A instanciação do módulo *CV* pode ser vista na Figura 6.

O módulo de visão computacional do EVA executa três tarefas que têm como base as imagens capturadas pela *PiCamera*, que fica acoplada sobre sua cabeça. O *driver* da câmera não permite que ela seja compartilhada por mais de um processo simultaneamente, fazendo com que as funcionalidades de identificação da face do usuário, de reconhecimento de expressões faciais e de leitura de *QR Code*, que fazem uso da câmera, tenham que ser implementadas em um só módulo, ou seja, no módulo que se conecta à câmera.

Um módulo pertencente ao grupo II, assim com os módulos do primeiro grupo, deve possuir as características básicas necessárias

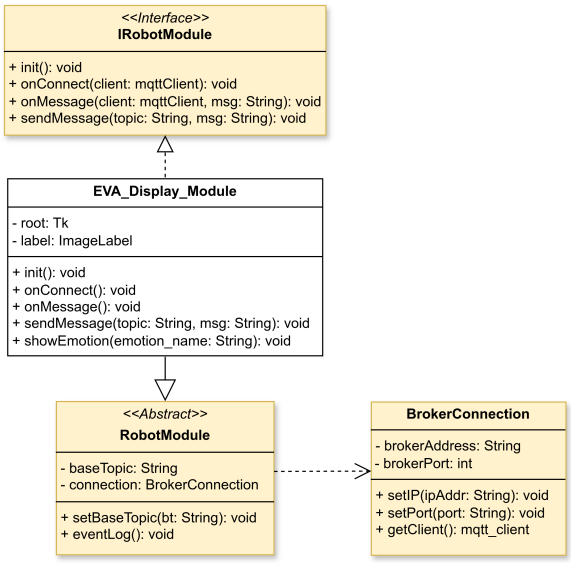

Figura 5: Instanciação do módulo *Display*.

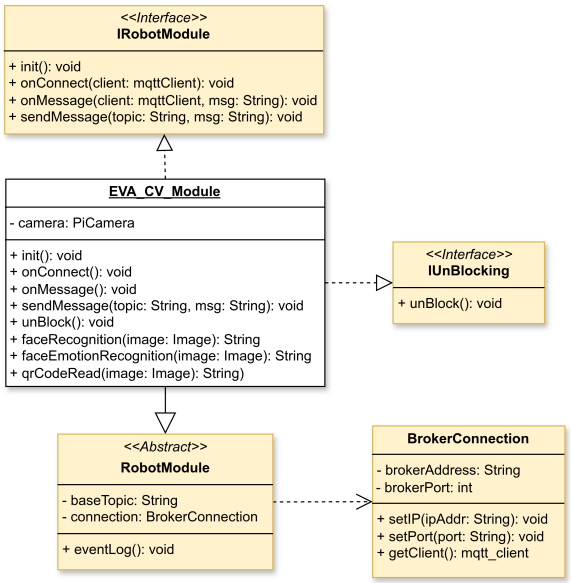

Figura 6: Instanciação do módulo CV (Visão Computacional).

para a implementação de uma funcionalidade no robô utilizando o *framework* proposto. Para isso, o módulo *CV* segue os mesmos passos de implementação dos módulos do grupo I. Os módulos do grupo II podem executar tarefas de maneira sincronizada com os outros módulos, sinalizando o término de sua atividade através de um mecanismo (uma mensagem) de desbloqueio. Para isso, o *framework* proposto disponibiliza uma interface chamada **IUnBlocking** que deve ser implementada pelo módulo. Esse método deve ser o responsável pelo envio (publicação) da mensagem que sinaliza o término da execução do módulo sendo implementado. Todos

Tabela 2: Tópicos e mensagens especiais utilizados no sistema de controle do EVA.

Propósito	Tópico	Descrição
Log de eventos	log	Os módulos do robô EVA utilizam este tópico para fornecer informações sobre seus estados e serviços.
Passagem de valores de variáveis	var	Este tópico é usado na passagem de valores de variáveis entre módulos. É utilizado pelos módulos CV, STT, TER e Script Player.
Sincronização entre módulos	state	Através deste tópico, os módulos podem funcionar de maneira síncrona, trocando mensagens de bloqueio e desbloqueio.

os módulos do EVA que implementam esse método enviam suas mensagens de sincronismo para um tópico especial, chamado *state*. No sistema de controle do EVA, todo módulo que controla ou é controlado por outro módulo utiliza a publicação mensagens nesse tópico como mecanismo de sincronização.

Além da utilização do mecanismo de sincronismo, com a implementação da interface **IUnBlocking**, um módulo do grupo II sempre retorna algum tipo de informação que é proveniente do seu serviço e essa informação é sempre no formato de texto (uma *string*). No sistema do EVA, os módulos que precisam passar suas informações para os outros módulos utilizam o método *sendMessage()*. Esse método é usado para enviar/publicar a informação a ser retornada em forma de mensagem através do tópico especial *var*. Sendo assim, qualquer módulo interessado em receber o valor de alguma variável deve assinar esse tópico. A Tabela 2 mostra, de uma maneira mais detalhada, a lista dos tópicos especiais utilizados no sistema do EVA.

4.2.3 ***Módulos - Grupo III.*** Os módulos do EVA que pertencem a este grupo são capazes de sincronizar sua execução com outros módulos. Eles implementam a interface **IUnBlocking** e são capazes de sinalizar o término de suas tarefas através do envio/publicação de mensagens no tópico especial, *state*. Porém, eles não retornam qualquer tipo de informação textual. Como pode ser visto na Tabela 1, os módulos deste grupo, são: o módulo *TTS*, o módulo *Audio* e o módule *Script Player*.

O módulo *TTS* é responsável por transformar um texto (uma *string*) em um arquivo de áudio. Esse arquivo contém o áudio do texto falado por uma das vozes disponibilizadas pelo serviço na nuvem do *IBM Watson*. O módulo é ativado a partir da recepção de uma mensagem publicada no tópico *talk*. Como pode ser visto na Tabela 1, a mensagem deve conter a especificação da voz a ser usada no processo de *TTS*, concatenada com o texto que deve ser transformado. Como pode ser visto na Figura 3, os arquivos de áudio gerados pelo *IBM Watson* são armazenados em uma pasta local chamada *TTS Cache Files*. Após o processo de geração do arquivo de áudio, o módulo *TTS* envia um comando de ativação para o módulo *Audio*, que deverá tocar o arquivo de áudio gerado pelo *Watson* e foi armazenado em *cache*. Esse comando é uma mensagem publicada no tópico *speech*, que é assinado pelo módulo *Audio* e contém o nome do arquivo de áudio com a fala.

O módulo *Audio* é responsável por tocar todo o tipo de áudio usado pelo robô, seja ele, um arquivo de música, um efeito sonoro ou um arquivo de áudio gerado pelo módulo *TTS*. Como pode ser visto na Tabela 1, o módulo assina dois tópicos, o tópico *sound* e o tópico *speech*, sendo ativado através do recebimento de mensagens publicadas nesses tópicos. O primeiro deles, o *sound*, recebe no texto da mensagem, o nome do arquivo de áudio a ser reproduzido, concatenado com uma informação *booleana* ("TRUE" ou "FALSE") que indica se o módulo deve executar de maneira síncrona ou assíncrona com o módulo de controle. Caso o valor passado seja verdadeiro, o módulo enviará um sinal de desbloqueio ao finalizar a execução do áudio. O segundo tópico assinado pelo módulo *Audio* é o *speech*. As mensagens recebidas nesse tópico ativam o módulo e contém apenas o nome do arquivo de áudio da fala, que foi previamente armazenado na pasta local *TTS Cache Files*. Todos os arquivos de áudio de fala são tocados de maneira síncrona, sinalizando o seu término através do tópico especial, *state*.

O módulo *Script Player* (*SP*) possui todas as características dos outros dois módulos do grupo III, porém, ele possui algumas características únicas. O módulo *SP* é um módulo de controle, sendo capaz de ler *scripts* de interação escritos na linguagem EvaML [7]. A EvaML é uma linguagem de programação *open-source* destinada ao desenvolvimento de *scripts* de interação para plataformas de robótica social. O módulo *SP* pode ler o conteúdo de um arquivo XML que contém o *script* EvaML e executar os comandos que representam as funcionalidades do robô, como: **<evaEmotion>**, **<light>**, **<motion>**, **<talk>** etc. Ao ler cada comando do *script* com seus respectivos atributos o módulo *SP* envia os sinais de ativação aos respectivos módulos fazendo com que o robô responda aos comandos sendo enviados. A fim de poder executar comandos de maneira sincronizada o módulo *SP* implementa uma outra interface fornecida pelo *framework*, a interface **IBlocking**. Através da implementação do método *block()* o módulo *SP* pode bloquear seu fluxo de execução, ativar um módulo que funcione de maneira síncrona como o módulo *STT* e, após a execução do módulo, que sinaliza o seu término enviando uma mensagem para o tópico especial *state*, o módulo *SP* é desbloqueado, seguindo seu fluxo executando o próximo comando.

Para facilitar a implementação dos módulos que precisam implementar algum tipo de memória para o robô, o *framework* disponibiliza uma classe abstrata que oferece um modelo simples de memória com alguns métodos concretos implementados. Esse módulo pode ter seus métodos sobrecarregados a fim de prover a manipulação de outros tipos de dados, além de *strings*. A Figura 7 mostra a interface **IBlocking**, a classe abstrata **RobotMemory** e o processo de instanciação dos três módulos do grupo III.

A Figura 8 apresenta um diagrama de sequência que demonstra como um comando de *TTS* é executado pelos módulos *SP* (a partir do seu envio), *TTS* (o processo de transformação de texto-para-fala) e *Audio* (execução do arquivo de áudio da fala). Através do acompanhamento da sequência apresentada no diagrama, é possível compreender com mais clareza o processo de sincronização entre módulos.

Ao executar um *script* EvaML e encontrar um comando do tipo **<talk>** o módulo *SP* publica uma mensagem no tópico *talk*. Como especificado na Tabela 1, a mensagem deve conter o timbre de voz que será utilizado pelo serviço do *IBM Watson* e o texto a ser transformado para fala. Em seguida, o módulo *SP* chama o seu método *block()*, bloqueando seu fluxo de execução. O módulo permanecerá

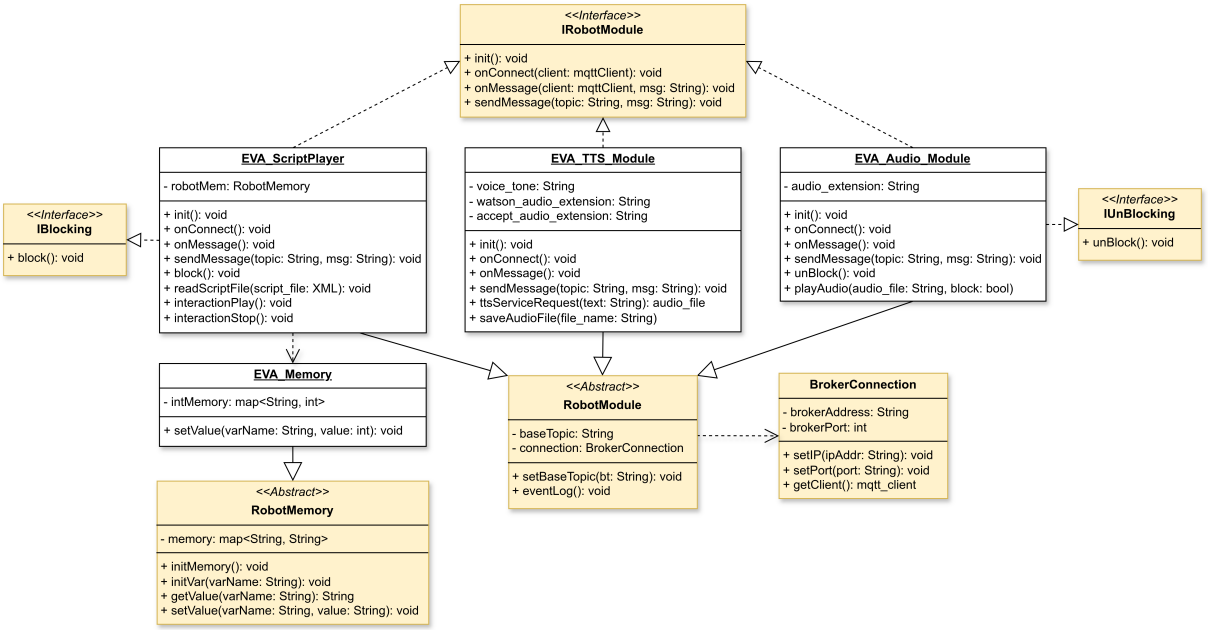

Figura 7: Instanciação dos módulos *TTS* (*Text-To-Speech*), *Audio* e *Script Player*.

bloqueado até que uma mensagem de desbloqueio seja publicada no tópico especial, *state*. Com o recebimento da mensagem, o módulo *TTS* é ativado, chamando logo em seguida, sua função interna que envia uma requisição para o serviço na nuvem do *IBM Watson*. Após a conversão do texto para fala, o módulo *TTS* salva o arquivo na pasta *TTS Cache Files*. O módulo *TTS* publica uma mensagem no tópico *speech* (do módulo *Audio)* contendo o nome do arquivo gerado e encerra sua execução. A mensagem é recebida pelo módulo *Audio*, que chama suas funções internas em sequência para ler o arquivo na pasta local e tocá-lo. Ao finalizar sua tarefa o módulo *Audio* envia um sinal de desbloqueio através da publicação no tópico especial, *state*. A mensagem é recebida pelo módulo *SP* fazendo com que ele seja desbloqueado, seguindo seu fluxo executando o próximo comando do *script*.

4.3 Funcionalidades do robô FRED

O robô FRED [9] (*Friendly Robot for EDucation and Healthcare*) é outra plataforma de robótica social *open-source*. Seu sistema de controle, assim como o do EVA, implementa suas capacidades de comunicação verbal e não verbal. O FRED pode falar (*TTS*), pode reconhecer a fala do usuário (*STT*), pode expressar emoções através do olhar, pode executar animações com os LEDs em seu tórax, pode tocar arquivos de áudio utilizando uma caixa de som externa, pode controlar efeitos sensoriais de luz usando uma lâmpada inteligente, pode classificar a emoção de um texto, pode fazer várias poses e se mover utilizando suas pernas, e tem os mesmos recursos de visão computacional do EVA. Sendo o FRED uma proposta de plataforma robótica social *open-source* de baixo custo, seus componentes de *hardware* são poucos e com poder de processamento restrito. Apesar disso, o FRED pode ser bastante expressivo através da combinação dos seus recursos de comunicação verbal e não verbal. Diferente

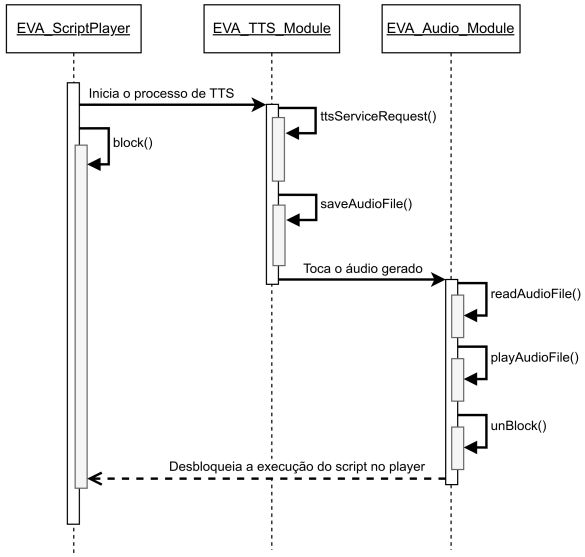

Figura 8: Diagrama de sequência para a execução do recurso de fala do robô EVA.

do EVA, que utiliza como base para o seu sistema de *software* um *Raspberry PI 4*, o FRED utiliza apenas uma placa Arduino UNO (que controla todo o seu *hardware* interno) e uma placa NodeMCU (rodando o *firmware* LUA). Para representar seus olhos e boca, o robô utiliza três matrizes de LED 8x8. Para o movimento de suas pernas são utilizados 4 servomotores SG-90. Todo o seu corpo é impresso em 3D e pode ser customizado. Ele mede cerca de 20cm de altura e pesa em torno de 200g. Seu custo total, incluindo o

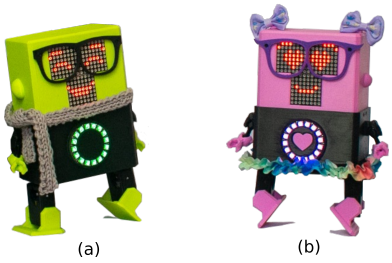

Figura 9: O robô FRED (a) e sua versão feminina Frida (b).

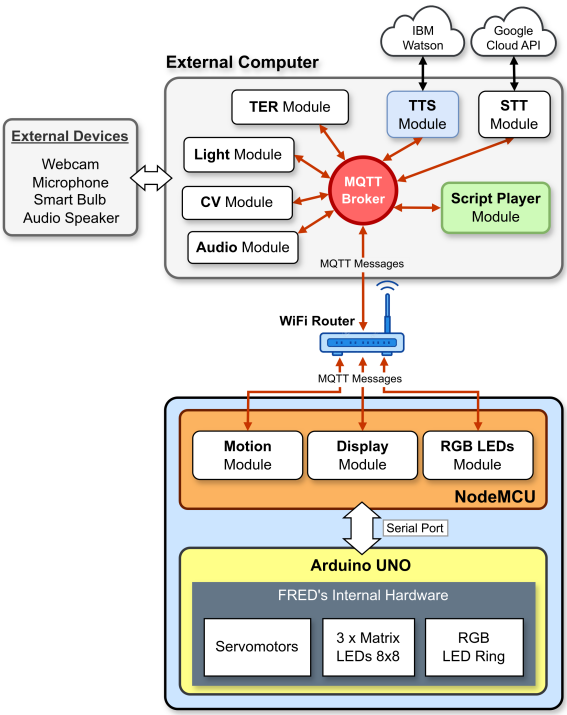

Figura 10: Arquitetura de *hardware* e *software* do FRED.

filamento PLA, fios, conectores e placas, fica em torno de US$ 70,00. Uma imagem do FRED e de sua versão feminina customizada, Frida, pode se vista na Figura 9.

4.4 Instanciação do *Software* do FRED

O sistema de controle do FRED foi implementado utilizando-se o *framework* proposto e conta com módulos similares ao do sistema do EVA. Esses módulos pertencem aos mesmos grupos I, II e III definidos anteriormente. A Figura 10 apresenta uma visão geral da arquitetura de *hardware* e *software* implementada para o FRED.

Assim como no sistema de controle do EVA, o módulo *Script Player* roda em um dispositivo externo ao robô como por exemplo um laptop. Porém, devido às capacidades de processamento restritas dos componentes de *hardware* do FRED, outros seis módulos rodam no mesmo dispositivo externo, são eles: *TER*, *TTS*, *STT*, *Light*, *CV* e *Audio*. As instanciações desses módulos em nada diferem dos mesmos módulos usados no *software* do EVA. Os mecanismos de sincronização entre os módulos e de transferência de valores de variáveis são os mesmos do sistema do EVA, utilizando os mesmos tópicos especiais, *state* e *var*, respectivamente. Os tópicos definidos para a ativação dos módulos são basicamente os mesmos apresentados na Tabela 1, com pequenas variações para funcionalidades especificas do FRED. Ele não pode mover sua cabeça e nem seus braços, mas pode se mover e fazer poses utilizando suas pernas. Portanto, os tópicos do módulo *Motion*, são: *pose* e *move*.

Como pode ser visto na Figura 10, no sistema do FRED, o *broker* roda num dispositivo externo (laptop). Os três módulos do FRED que não rodam no dispositivo externo, são: o *Motion*, o *Display* e o *RGB LEDs*. Eles são implementados numa placa NodeMCU, rodando um *firmware* LUA, permitindo ao FRED se tornar um cliente MQTT. A placa NodeMCU, além de permitir a conexão com *broker*, permite a conexão e o controle da placa Arduino através de uma conexão com a porta serial.

A instanciação dos três módulos segue o mesmo padrão descrito para os módulos do *Grupo I*, na Seção 4.2. Os módulos desse grupo não executam de maneira síncrona com outros módulos, isto é, não sinalizam o final das suas tarefas. Além disso, eles não retornam qualquer tipo de informação em forma de texto. Um vídeo dos robôs funcionado com o sistema de *software* desenvolvido com o *framework* proposto pode ser visto neste link.

5 CONCLUSÃO

Este trabalho propôs um *framework* para desenvolvimento de sistemas de controle para plataformas de robótica social. O *framework* proposto adotou uma abordagem de especificação orientada a objetos com o intuito de facilitar o entendimento de sua estruturação. O *framework* utiliza um paradigma *publish-subscribe* para troca mensagens assíncronas entre os módulos desenvolvidos tendo como dependência apenas a instalação de um *broker* MQTT. Foram apresentados os elementos que compõem o *framework*, suas interfaces e suas classes concretas e abstratas. A utilização do *framework* foi validada através de dois cenários concretos utilizando as plataformas de robótica social EVA e FRED.

Uma das limitações do *framework* proposto pode estar na utilização do paradigma de troca de mensagens assíncronas *publish-subscribe* pois a comunicação entre os módulos baseada na troca de mensagens usando MQTT pode introduzir alguma latência, o que pode ser inadequado para aplicações de tempo real muito rigorosas. Outro ponto que merece atenção é o uso seguro do MQTT para evitar que potenciais invasores publiquem mensagens no *broker*, atrapalhando o funcionamento dos robôs. A solução segura será desenvolvida futuramente.

Como trabalhos futuros, pretende-se realizar estudos de desempenho comparativos entre o *framework* proposto e outros *frameworks* populares como ROS, NAOqi, YARP e OROCOS. Pretende-se também demonstrar a aplicação prática do *framework* proposto em diferentes contextos como educação e saúde.

AGRADECIMENTOS

Os autores agradecem o apoio recebido do Google Research, CAPES, CAPES PRINT, CNPq, FINEP, INCT-MACC, ICNT-ICoNIoT e FAPERJ.

REFERÊNCIAS

[1] Michel Albonico, Milica Đorđević, Engel Hamer, and Ivano Malavolta. 2023. Software engineering research on the Robot Operating System: A systematic mapping study. *Journal of Systems and Software* 197 (2023), 111574.

[2] J.-C. Baillie. 2004. URBI: towards a universal robotic body interface. In *4th IEEE/RAS International Conference on Humanoid Robots, 2004.*, Vol. 1. 33–51 Vol. 1. https://doi.org/10.1109/ICHR.2004.1442112

[3] Herman Bruyninckx. 2002. OROCOS: design and implementation of a robot control software framework. In *Proceedings of IEEE International Conference on Robotics and Automation*. Citeseer.

[4] Paulo Canelas, Miguel Tavares, Ricardo Cordeiro, Alcides Fonseca, and Christopher S Timperley. 2022. An experience report on challenges in learning the robot operating system. In *Proceedings of the 4th International Workshop on Robotics Software Engineering*. 33–38.

[5] Dagoberto Cruz-Sandoval and Jesús Favela. 2017. Semi-autonomous conversational robot to deal with problematic behaviors from people with dementia. In *Ubiquitous Computing and Ambient Intelligence: 11th International Conference, UCAmI 2017, Philadelphia, PA, USA, November 7–10, 2017, Proceedings*. Springer, 677–688.

[6] Dagoberto Cruz-Sandoval, Arturo Morales-Tellez, Eduardo Benitez Sandoval, and Jesus Favela. 2020. A social robot as therapy facilitator in interventions to deal with dementia-related behavioral symptoms. In *Proceedings of the 2020 ACM/IEEE international conference on human-robot interaction*. 161–169.

[7] Marcelo Marques da Rocha and Débora Christina Muchaluat Saade. 2023. Embodied Voice Assistant Markup Language. In *Proceedings of the 29th Brazilian Symposium on Multimedia and the Web*. 246–254.

[8] Phong Ba Dao. 2021. OROMACS: A design framework for multi-agent control system. *International Journal of Control, Automation and Systems* 19, 5 (2021), 1907–1919.

[9] Jesus Favela, Dagoberto Cruz-Sandoval, Marcelo Marques da Rocha, and Débora Christina Muchaluat-Saade. 2024. Social Robots for Healthcare and Education in Latin America. *Commun. ACM* 67, 8 (2024), 70–71.

[10] Carlos Solon Soares Guimarães Júnior. 2015. Proposta de um framework baseado em arquitetura orientada a serviços para a robótica. (2015).

[11] Anna Henschel, Guy Laban, and Emily S Cross. 2020. What Makes a Robot Social? A Review of Social Robots from Fiction to a Home or Hospital Near You. (2020).

[12] Heiko Koziolek, Sten Grüner, and Julius Rückert. 2020. A Comparison of MQTT Brokers for Distributed IoT Edge Computing. In *Software Architecture*, Anton Jansen, Ivano Malavolta, Henry Muccini, Ipek Ozkaya, and Olaf Zimmermann (Eds.). Springer International Publishing, Cham, 352–368.

[13] Marcelo Marques da Rocha, Dagoberto Cruz-Sandoval, Jesus Favela, and Débora C Muchaluat-Saade. 2024. Design and usability evaluation of the EvaSIM simulator for a socially assistive robot. *Multimedia Tools and Applications* (2024), 1–26.

[14] Jon Martin, Ander Ansuategi, Iñaki Maurtua, Aitor Gutierrez, David Obregón, Oskar Casquero, and Marga Marcos. 2021. A generic ROS-based control architecture for pest inspection and treatment in greenhouses using a mobile manipulator. *IEEE access* 9 (2021), 94981–94995.

[15] Giorgio Metta, Paul Fitzpatrick, and Lorenzo Natale. 2006. YARP: yet another robot platform. *International Journal of Advanced Robotic Systems* 3, 1 (2006), 8.

[16] Roger S Pressman and Bruce R Maxim. 2021. *Engenharia de software-9*. McGraw Hill Brasil.

[17] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. 2009. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, Vol. 3. Kobe, Japan, 5.

[18] Hannes Ritschel. 2022. Real-time generation and adaptation of social companion robot behaviors. (2022).

[19] Hannes Ritschel, Thomas Kiderle, and Elisabeth André. 2021. Implementing Parallel and Independent Movements for a Social Robot's Affective Expressions. In *2021 9th International Conference on Affective Computing and Intelligent Interaction Workshops and Demos (ACIIW)*. IEEE, 1–4.

[20] Bruno Siciliano and Oussama Khatib. 2016. Robotics and the Handbook. In *Springer Handbook of Robotics*. Springer, 1–6.

[21] Anushka Subedi, Dipesh Pandey, and Deepti Mishra. 2021. Programming Nao as an educational agent: a comparison between Choregraphe and Python SDK. In *The Proceedings of the International Conference on Smart City Applications*. Springer, 367–377.

[22] Veronika Szücs, György Károlyi, András Tatár, and Attila Magyar. 2018. Voice Controlled Humanoid Robot based Movement Rehabilitation Framework. In *2018 9th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)*. 000191–000196. https://doi.org/10.1109/CogInfoCom.2018.8639704