UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de Bauru

MICHEL DE MATTOS FERNANDES

Controle de Processos como um Serviço: Uma Abordagem para a Indústria 4.0

Bauru
2020

MICHEL DE MATTOS FERNANDES

CONTROLE DE PROCESSOS COMO UM SERVIÇO: UMA ABORDAGEM PARA A INDÚSTRIA 4.0

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Faculdade de Engenharia da Universidade Estadual Paulista "Júlio de Mesquita Filho" para obtenção do título de Mestre em Engenharia Elétrica.

Área de Concentração: Automação

Linha de Pesquisa: Mecatrônica

Orientador: Prof. Dr. Eduardo Paciência Godoy

Bauru 2020

F363c

Fernandes, Michel de Mattos
Controle de processos como um serviço: uma abordagem para a indústria 4.0 / Michel de Mattos Fernandes. -- Bauru, 2020
87 p. : il., tabs.

Dissertação (mestrado) - Universidade Estadual Paulista (Unesp), Faculdade de Engenharia, Bauru
Orientador: Eduardo Paciência Godoy

1. Arquitetura Orientada a Microserviços (MOA). 2. Framework Moleculer. 3. Internet das Coisas Industrial (IIoT). 4. Sistema de Controle via Rede. 5. Eletrônica Embarcada. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca da Faculdade de Engenharia, Bauru. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

ATA DA DEFESA PÚBLICA DA DISSERTAÇÃO DE MESTRADO DE MICHEL DE MATTOS FERNANDES, DISCENTE DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA, DA FACULDADE DE ENGENHARIA - CÂMPUS DE BAURU.

Aos 04 dias do mês de novembro do ano de 2020, às 09:00 horas, por meio de Videoconferência, realizou-se a defesa de DISSERTAÇÃO DE MESTRADO de MICHEL DE MATTOS FERNANDES, intitulada **CONTROLE DE PROCESSOS COMO UM SERVIÇO: UMA ABORDAGEM PARA A INDÚSTRIA 4.0**. A Comissão Examinadora foi constituida pelos seguintes membros: Prof. Dr. EDUARDO PACIÊNCIA GODOY (Orientador(a) - Participação Virtual) do(a) Departamento de Engenharia de Controle e Automação / Instituto de Ciência e Tecnologia - UNESP - Câmpus de Sorocaba, Prof. Dr. ANDRE LUIZ ANDREOLI (Participação Virtual) do(a) Departamento de Engenharia Elétrica / Faculdade de Engenharia de Bauru - UNESP, Prof. Dr. BRUNO AUGUSTO ANGÉLICO (Participação Virtual) do(a) Departamento de Engenharia de Telecomunicações e Controle / Escola Politécnica da USP. Após a exposição pelo mestrando e arguição pelos membros da Comissão Examinadora que participaram do ato, de forma presencial e/ou virtual, o discente recebeu o conceito final: **APROVADO**. Nada mais havendo, foi lavrada a presente ata, que após lida e aprovada, foi assinada pelo(a) Presidente(a) da Comissão Examinadora.

Prof. Dr. EDUARDO PACIÊNCIA GODOY

"A menos que modifiquemos a nossa maneira de pensar, não seremos capazes de resolver os problemas causados pela forma como nos acostumamos a ver o mundo"

(Albert Einstein)

À Deus, aos meus pais que sempre me apoiaram e me incentivaram, aos meus familiares e amigos

AGRADECIMENTOS

À Deus por me dar capacidade e forças em todos os momentos da minha vida.

À minha família, em especial minha mãe Tereza e meu pai Oziel, que se dedicaram para que eu pudesse estudar, me ajudando no que fosse preciso.

À Yoko Arimoto que me incentivou e apoiou durante esta etapa da minha vida.

Ao meu orientador Prof. Dr. Eduardo Paciência Godoy por todos os ensinamentos, amizade e dedicação. É um professor que além de possuir um enorme conhecimento nos inspira a sermos cada vez melhores no que fazemos.

Aos meus colegas de pós-graduação que me ajudaram, em especial ao Jeferson André Bigheti, que além de um grande amigo também foi um professor.

À Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) pelo apoio para o desenvolvimento desse trabalho: Processo 2018/19984-4.

RESUMO

A necessidade de viabilizar e padronizar o acesso às informações do chão de fábrica pelos sistemas de gerenciamento e tomada de decisões têm norteado o desenvolvimento de tecnologias flexíveis e adaptáveis às mudanças da indústria. A Indústria 4.0 (I4.0) surge como a evolução dos sistemas produtivos através da interconexão e integração entre sistemas e equipamentos, promovendo a disponibilização das informações necessárias para as melhores decisões de produção e gerenciamento industrial. Para o desenvolvimento da I4.0, é necessária a aplicação de tecnologias recentes como a Internet das Coisas Industrial (IIoT), os Sistemas de Controle via Redes, a Computação em Nuvem, entre outras. O grande desafio da I4.0 é promover a integração entre essas tecnologias, equipamentos e sistemas alocados em diferentes níveis hierárquicos dos sistemas industriais. Nesse sentido, um paradigma recente tem sido o da automação colaborativa por meio do uso e compartilhamento de serviços para obtenção de uma arquitetura flexível, distribuída e totalmente integrada através de redes de comunicação. Diante desse contexto, este trabalho desenvolveu uma nova abordagem para o controle de processos industriais com foco na I4.0. O objetivo foi o estudo e implementação de controle de processos como um serviço. Essa nova arquitetura de controle orientada a serviços é baseada no uso do *framework* Moleculer e plataforma *Node.js*. Os serviços desenvolvidos foram: o serviço Controlador Lógico Programável (CLP), o serviço de Aquisição de Dados (DAQ) e o serviço de Controle PIDPlus, os quais podem ser executados em diferentes plataformas como computadores, sistemas embarcados e em nuvem. Uma descrição do desenvolvimento dos serviços DAQ, Controle PIDPlus e CLP, bem como da composição desses serviços por orquestração para criação de uma aplicação de Controle e Supervisão são apresentados. Para validação experimental, uma malha de controle de um motor de corrente contínua e a automação discreta de uma planta industrial virtual usando os serviços foram implementadas. Resultados relacionados ao desempenho de controle e tempo de resposta da malha de controle demonstram a viabilidade da implementação do controle de processos como um serviço. Os resultados também demonstram que essa nova abordagem de controle de processos e automação fornece modularidade, escalabilidade e interoperabilidade para aplicações de controle com foco na I4.0.

Palavras-chave: Serviços, *Framework* Moleculer, Internet das Coisas Industrial, Manufatura Avançada.

ABSTRACT

The need to enable and standardize access to the factory floor information by management and decision-making systems has guided the development of flexible technologies that are adaptable to evolution of industry. Industry 4.0 (I4.0) emerges as the evolution of productive systems through the interconnection and integration between systems and equipment, promoting the availability of necessary information for the best production decisions and industrial management. For the development of I4.0, it is necessary to apply recent technologies such as the Industrial Internet of Things (IIoT), Network Control Systems, Cloud Computing, among others. The major challenge of I4.0 is to promote the integration among these technologies, equipment and systems allocated at different hierarchical levels of industrial systems. In this sense, a recent paradigm has been the collaborative automation through the use and sharing of services to achieve a flexible, distributed and fully integrated networked architecture. In this context, this work developed a new approach for the control of industrial processes with a focus on I4.0. The objective was the study and implementation of process control as a service. This new service-oriented control architecture is based on the use of the Moleculer framework and Node.js platform. The services developed were: the Programmable Logic Controller (PLC) service, the Data Acquisition service (DAQ) and the PIDPlus Control service, which can be run on different platforms such as computers, embedded systems, and in cloud. A description of the development of the DAQ, PIDPlus Control and PLC services, as well as the composition of these services by orchestration for creating a Control and Supervision application are presented. For experimental validation, a control loop for a direct current motor and the discrete automation of a virtual industrial plant using the services were implemented. Results related to control performance and control response time demonstrate the feasibility of implementing processes control as services. The results also demonstrate that this new approach to process control and automation provides modularity, scalability and interoperability for control applications with a focus on I4.0.

Keywords: Services; Moleculer Framework; Industrial Internet of Things; Advanced Manufacturing.

LISTA DE FIGURAS

Figura 1 – As Revoluções Industriais. 12
Figura 2 – Integração de Processos na Industria 4.0 através de Tecnologias de IoT e Serviços. 13
Figura 3 – Sistema de Controle em Nuvem: (a) Redundância de Controladores, (b) Localização de Controladores em Nuvem. 14
Figura 4 – Arquitetura SOA Aplicada na Automação de um Processo. 15
Figura 5 – Representação das intersecções entre IoT, IIoT, CPS e Indústria 4.0. 17
Figura 6 – Comparativo entre orquestração e coreografia de serviços. 21
Figura 7 – Esquemático da composição dos microserviços. 23
Figura 8 – Arquitetura típica de microserviços. 24
Figura 9 – Arquitetura Orientada a Microserviços do Moleculer. 27
Figura 10 – Exemplo de Interconexão de Serviços do Moleculer. 28
Figura 11 – Estrutura do *ServiceBroker*. 31
Figura 12 – Configuração padrão do *ServiceBroker*. 31
Figura 13 – Configuração personalizada do *ServiceBroker*. 31
Figura 14 – Modelo de microserviço do Moleculer. 32
Figura 15 – Esquema de serviço simples para definir duas ações. 33
Figura 16 – Método de comunicação RPC das chamadas externas de ações. 33
Figura 17 – Exemplo de chamada de ação "*math*" via *broker.call* de um microserviço do Moleculer. 34
Figura 18 – Exemplo da propriedade *Methods* de um microserviço do Moleculer. 35
Figura 19 – Diagrama de evento balanceado para microserviço no Moleculer. 36
Figura 20 – Exemplo da propriedade do grupo do método evento de um microserviço do Moleculer. 36
Figura 21 – Exemplo da função *broker.emit* de um microserviço do Moleculer. 37
Figura 22 – Exemplo do método *broker.broadcast* de um microserviço do Moleculer. 37
Figura 23 – Diagrama de um evento *Broadcast* de um microserviço no Moleculer. 37
Figura 24 – Exemplo de configuração de um *ServiceBroker* com registrador de *logs*. 38
Figura 25 – Microserviço *Transporter* da Arquitetura MOA. 39
Figura 26 – Configuração do REPL no Moleculer 40
Figura 27 – Opções do REPL no Moleculer 40
Figura 28 – Diagrama da Arquitetura MOA com *framework* Moleculer para Aplicações de IIoT e I4.0. 41

Figura 29 – Microserviço DAQ. 43
Figura 30 – Estrutura de Implementação do Nó e Serviço DAQ. 43
Figura 31 – Microserviço Controle PIDPlus. 45
Figura 32 – Estrutura do controlador PIDPlus. 45
Figura 33 – Microserviço CLP. 47
Figura 34 – Aplicação Interna do Moleculer por Orquestração de Serviços. 49
Figura 35 – Aplicação Externa no Moleculer por Orquestração de Serviços. 50
Figura 36 – Sistema DAQ. 52
Figura 37 – Trecho de código JavaScript de um microserviço do Moleculer. 53
Figura 38 – Esquema de ligação para testes do microserviço DAQ 54
Figura 39 – Estrutura de Implementação do Nó e Serviço Controle PIDPlus. 55
Figura 40 – Esquema de ligação simplificado do controle em malha fechada 57
Figura 41 – Estrutura do Experimento de Controle de Processo Usando Serviços com Planta Didática. 58
Figura 42 – Resposta do Experimento de Controle de Processo usando Serviços. 59
Figura 43 – Sinal de Controle do Experimento de Controle de Processo usando Serviços. 60
Figura 44 – Histograma do Experimento de Controle usando Serviços - Tempo da Orquestração 61
Figura 45 – Estrutura de implementação do microserviço CLP 63
Figura 46 – Placa UniPi 1.1 64
Figura 47 – Esquema de ligação para o teste do microserviço CLP 65
Figura 48 – Esquema simplificado da planta de controle em malha fechada 66
Figura 49 – Planta Industrial em Simulação no *software* Factory I/O 67
Figura 50 – MOA da aplicação do microserviço CLP 68
Figura 51 – Estrutura de *hardware* e *software* para experimento do CLP como um Serviço 69
Figura 52 – *Dashboard* de monitoramento e controle desenvolvido no Node-RED 70
Figura 53 – Fluxo para supervisão das entradas digitais construído no Node-RED 80
Figura 54 – Parâmetros do bloco *inject* Node-RED 81
Figura 55 – Configuração do Broker do Moleculer no Node-RED 81
Figura 56 – Configuração de acesso à ações do Moleculer através do Node-RED 82
Figura 57 – *Dashboard* desenvolvido no Node-RED 83

LISTA DE TABELAS

Tabela 1 – Experimento de Análise de Tempo para o Controle como um Serviço 62

LISTA DE SIGLAS

AC	*Alternating Current* (Corrente Alternada)
ADC	*Analog Digital Converter* (Conversor Analógico Digital)
AMQP	*Advanced Message Queuing Protocol* (Protocolo Avançado de Filas de Mensagem)
API	*Application-Programming Interface* (Interface de Programação de Aplicativos)
ARUM	*Adaptive Production Management* (Gerenciamento Adaptativo de Produção)
AS-i	*Actuator Sensor Interface* (Interface Sensor Atuador)
CAN	*Controller Area Network*
CC	Corrente Contínua
CLP	Controlador Lógico Programável
CNC	*Computer Numeric Control* (Controle Numérico Computadorizado)
CoAP	*Constrained Application Protocol* (Protocolo de Aplicação Restrita)
CPS	*Cyber Physical Systems* (Sistemas Cyber Físicos)
CSS	*Cascading Style Sheets* (Folha de Estilo em Cascatas)
DAQ	*Data Acquisition* (Aquisição de Dados)
DC	*Direct Current* (Corrente Contínua)
DCS	*Distributed Control System* (Sistemas de Controle Distribuído)
DPWS	*Device Profile for Web Services* (Perfil de Dispositivos para Serviços *Web*)
ERP	*Enterprise Resource Planning* (Planejamento de Recursos Empresariais)
FAPESP	Fundação de Amparo à Pesquisa do Estado de São Paulo
FBD	*Function Block Diagram* (Diagrama de Blocos Funcionais)
GPIO	*General Purpose Input/Output* (Entrada/Saída de Uso Geral)
GRACE	*inteGration of pRocess and quAlity Control using multi-agEnt technology* (Integração do controle de processos e qualidade usando a tecnologia multi-agent)
HTTP	*Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto)
I/O	*In/Out* (Entradas/Saídas)
I2C	*Inter-Integrated Circuit* (Circuito Inter-Integrado)
I4.0	Indústria 4.0
IEC	*International Electrotechnical Commission* (Comissão Eletrotécnica Internacional)
IIoT	*Industrial Internet of Things* (Internet das Coisas Industrial)
IL	*Instruction List* (Lista de Instruções)

IMC-AESOP	*Architecture for Service-Oriented Process Monitoring and Control* (Arquitetura para Monitoramento e Controle de Processos Orientados a Serviços)
IoT	*Internet of Things* (Internet das Coisas)
IP	*Internet Protocol* (Protocolo de Internet)
IPV6	*Internet Protocol Version 6* (Protocolo de Internet Versão 6)
ISA	*International Society of Automation* (Sociedade Internacional de Automação)
JSON	*JavaScript Object Notation* (Notação de Objetos JavaScript)
LD	*Ladder Diagram* (Diagrama *Ladder*)
LED	*Light Emitting Diode* (Diodo Emissor de Luz)
M2M	*Machine-to-Machine* (Máquina a Máquina)
MES	*Manufacturing Execution System* (Sistema de Execução de Manufatura)
MOA	*Microservices Oriented Architecture* (Arquitetura Orientada a Microserviços)
MQTT	*Message Queuing Telemetry Transport* (Transmissão de Telemetria de Espera de Mensagens)
MV	*Manipulated Variable* (Variável Manipulada)
NCS	*Networked Control Systems* (Sistema de Controle em Rede)
NPM	*Node Package Manager* (Gerenciador de Pacotes do Node)
OPC UA	*Open Platform Communications Unified Architecture* (Plataforma Aberta de Comunicação – Arquitetura Unificada)
PaaS	*Platform as a Service* (Plataforma como serviço)
PC	*Pessoal Computer* (Computador Pessoal)
PCI	*Peripheral Component Interconnect* (Interconector de Componentes Periféricos)
PID	*Proportional Integral Derivative* (Proporcional Integral Derivativo)
PLC	*Programmable Logic Controller* (Controlador Lógico Programável)
PV	*Process Variable* (Variável de Processo)
PWM	*Pulse Width Modulation* (Modulação Por Largura De Pulso)
RaaS	*Robot as a Service* (Robô como serviço)
REPL	*Read-Eval-Print Loop* (Loop de Leitura, Avaliação e Impressão)
REST	*Representational State Transfer* (Transferência Representacional de Estado)
RFID	*Radio-Frequency Identification* (Identificação por radiofrequência)
RPC	*Remote Procedure Call* (Chamada Remota de Procedimento)
RPM	Rotações Por Minuto
RTC	*Real-Time Clock* (Relógio De Tempo Real)

SCADA	*Supervisory Control and Data Acquisition* (Controle de Supervisão e Aquisição de Dados)
SFC	*Sequential Function Chart* (Sequenciamento Gráfico de Funções)
SOA	*Service Oriented Architecture* (Arquitetura Orientada a Serviço)
SOCRADES	*Service-oriented cross-layer infrastructure for distributed smart embedded systems* (Infraestrutura de camada cruzada orientada a serviços para sistemas embarcados inteligentes distribuídos)
ST	*Structured Text* (Texto Estruturado)
TA	Tecnologia de Automação
TCP	*Transmission Control Protocol* (Protocolo de Controle de Transmissão)
TI	Tecnologia da Informação
TIC	Tecnologias de Informação e Comunicação
UART	*Universal Asynchrounous Receiver/Transmiter* (Receptor/Transmissor Universal Assíncrono)
UDP	*User Datagram Protocol* (Protocolo de datagrama do usuário)
URL	*Uniform Resource Locator* (Localizador Padrão de Recursos)
USB	*Universal Serial Bus* (Barramento Serial Universal)
VDC	*Voltage Direct Current* (Tensão Corrente Contínua)
XML	*Extensible Markup Language* (Linguagem de Marcação Estendida)

SUMÁRIO

1 INTRODUÇÃO E JUSTIFICATIVA 12

1.1 JUSTIFICATIVA 12

1.2 OBJETIVO 15

1.3 ESTRUTURA DO TRABALHO 16

2 REVISÃO BIBLIOGRÁFICA 17

2.1 INDÚSTRIA 4.0 17

2.2 ARQUITETURA ORIENTADA A SERVIÇOS (SOA) 19

2.2.1 Composição de Serviços 20

2.2.2 Microserviços 22

2.3 APLICAÇÕES INDUSTRIAIS USANDO SERVIÇOS 24

3 *FRAMEWORK* MOLECULER 27

3.1 *SERVICE BROKER* 30

3.2 SERVIÇO (*SERVICE*) 31

3.3 *GATEWAY* (API) 38

3.4 *TRANSPORTER* 39

3.5 REPL CONSOLE 40

4 PROPOSTA DO TRABALHO 41

4.1 MICROSERVIÇO DAQ 42

4.2 MICROSERVIÇO CONTROLE PIDPLUS 44

4.2.1 Algoritmo de Controle PIDPlus 45

4.3 MICROSERVIÇO CLP 47

4.4 APLICAÇÕES INTERNAS E EXTERNAS 48

5 DESENVOLVIMENTO DOS SERVIÇOS E RESULTADOS 51

5.1 DESENVOLVIMENTO DO MICROSERVIÇO DAQ 51

5.2 DESENVOLVIMENTO DO MICROSERVIÇO CONTROLE PIDPLUS 55

5.3 APLICAÇÃO EXTERNA DE CONTROLE DE PROCESSO USANDO SERVIÇOS DAQ E CONTROLE PIDPLUS 56

5.4 DESENVOLVIMENTO DO MICROSERVIÇO CLP 62

5.5 APLICAÇÃO INTERNA DE MONITORAMENTO, CONTROLE E AUTOMAÇÃO DE PROCESSOS USANDO O MICROSERVIÇO CLP 65

5.6 DISCUSSÕES E CONSIDERAÇÕES FINAIS 70

6 CONCLUSÕES **73**

REFERÊNCIAS **75**

APÊNDICES **80**

CRIAÇÃO DE UMA APLICAÇÃO INTERNA NO NODE-RED 80

1 INTRODUÇÃO E JUSTIFICATIVA

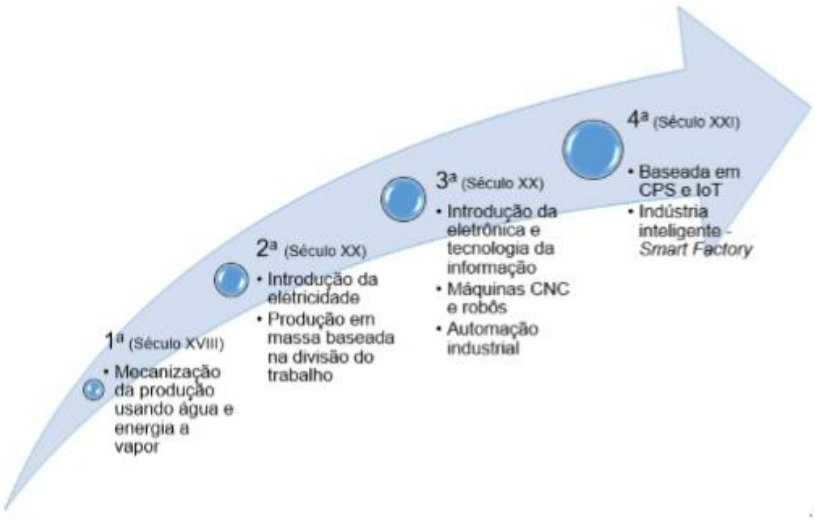
1.1 Justificativa

As tecnologias de automação (TA) tornaram-se essenciais para melhorar os processos industriais, graças à sua alta capacidade em acelerar processos de produção e manufatura, obtendo assim uma maior eficiência e qualidade dentro de menores custos e tempos. Dessa forma, a indústria tem evoluído de forma a incorporar novas tecnologias e consequentemente obter maior produtividade (SAUTER et al., 2011; BANGEMANN et al., 2014).

A evolução e o desenvolvimento das tecnologias digitais contribuíram para a criação de novos métodos de produção industrial baseados na automação, robótica, inteligência artificial, Internet das Coisas (IoT – *Internet of Things*) e inteligência de dados, dentre outras inovações. A utilização dessas tecnologias no contexto industrial, empregadas de modo a permitir competitividade aos negócios, otimizando a eficiência da cadeia produtiva, adicionando valor aos produtos, racionalizando o uso dos recursos e customizando as soluções tecnológicas é denominada Indústria 4.0 (I4.0).

A I4.0, também chamada de Quarta Revolução Industrial, é um novo conceito que representa uma evolução dos sistemas produtivos atuais a partir da convergência entre novas tecnologias de automação (TA) e tecnologia da informação (TI) (BLANCHET et al., 2014; LU, 2017) como pode ser visto na Figura 1.

Figura 1 – As Revoluções Industriais.



Fonte: Pisching (2017).

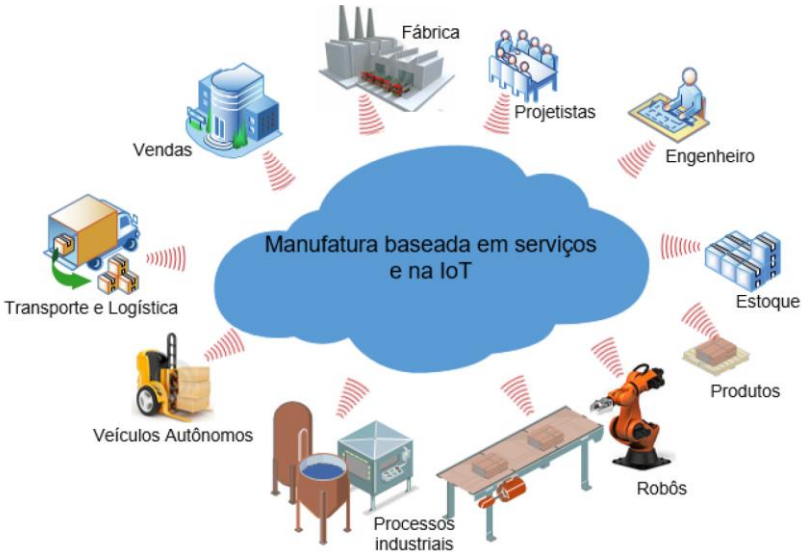
No contexto da I4.0, os sistemas de controle rígidos e centralizados das fábricas dão lugar para inteligência descentralizada, em um ambiente onde todos os equipamentos e máquinas

estão conectadas em redes e disponibilizando informações de forma única (COLOMBO; KARNOUSKOS; BANGEMANN, 2014). Grande parte das tecnologias necessárias para a implementação da I4.0 já existem atualmente e encontram-se em diferentes estágios de desenvolvimento. Entre elas podem-se citar (HEGAZY; HEFEEDA, 2015; LEE; BAGHERI; KAO, 2015; RAJKUMAR et al., 2010; STANKOVIC, 2014; ZUEHLKE, 2010):

- ✓ Protocolo IPV6 para ampliação dos pontos de conexão IP a todos os equipamentos;
- ✓ Ampliação das redes sem fio para flexibilização da comunicação e aquisição de dados;
- ✓ Sistemas de controle via redes (NCS – *Networked Control Systems*) e sistemas *cyber*-físicos (CPS – *Cyber Physical Systems*);
- ✓ RFID para rastreamento e identificação de materiais e suas informações;
- ✓ Uso de virtualização de sistemas e serviços;
- ✓ Internet das Coisas Industrial (IIoT – *Industrial Internet of Things*) e Computação em nuvem (*Cloud Computing*) visando a conexão integrada e interoperável de todos os equipamentos em redes e o compartilhamento das informações na nuvem;
- ✓ *Big Data* onde todas as informações são reunidas e utilizadas para tomada de decisões.

O grande desafio da Industria 4.0 é a integração entre tecnologias existentes, para que seja possível obter uma nova realidade produtiva, onde tudo poderá estar conectado permitindo que melhores decisões de produção, custo e segurança sejam tomadas, tudo sob demanda e em tempo real. Nesse sentido, muitos trabalhos estão sendo desenvolvidos buscando soluções que agreguem essas tecnologias e permitam a integração dos processos produtivos e equipamentos de automação a serviços de TI que estão armazenados na nuvem, conforme Figura 2.

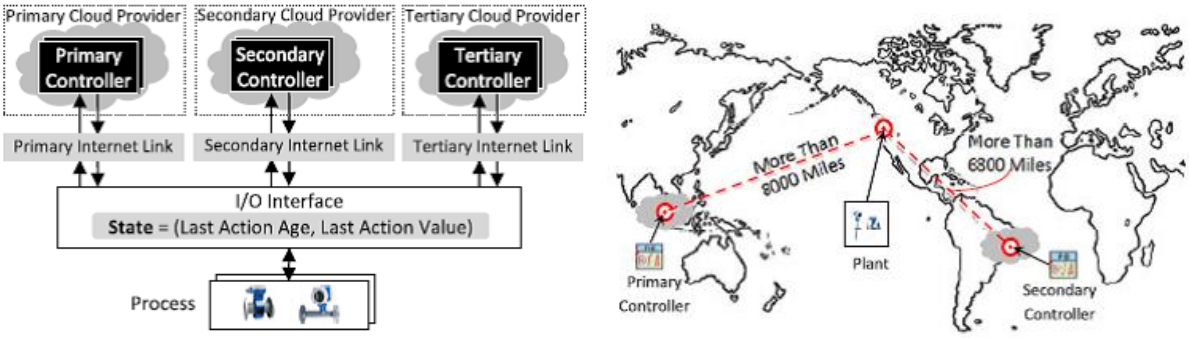
Figura 2 – Integração de Processos na Industria 4.0 através de Tecnologias de IoT e Serviços.



Fonte: Pisching (2018).

Hegazy e Heffeda (2015) apresentam um novo serviço na nuvem, “automação industrial como serviço”, onde os controladores são hospedados em dois servidores em nuvem, fisicamente separados, conforme apresentado na Figura 3. Nesta proposta diversas aplicações como controle em malha fechada e virtualização de controladores são realizadas na nuvem. De acordo com os autores, a proposta traria inúmeros benefícios como redução de custos e tempo de desenvolvimento das aplicações utilizando a virtualização de controladores, além da redução de *hardware* utilizado. O trabalho apresenta uma implementação de redundância em nuvem de controladores industriais e um sistema de controle de uma planta utilizando a nuvem comercial da *Amazon Web Services*.

Figura 3 – Sistema de Controle em Nuvem: (a) Redundância de Controladores, (b) Localização de Controladores em Nuvem.



Fonte: Hegazy & Hefeeda (2015).

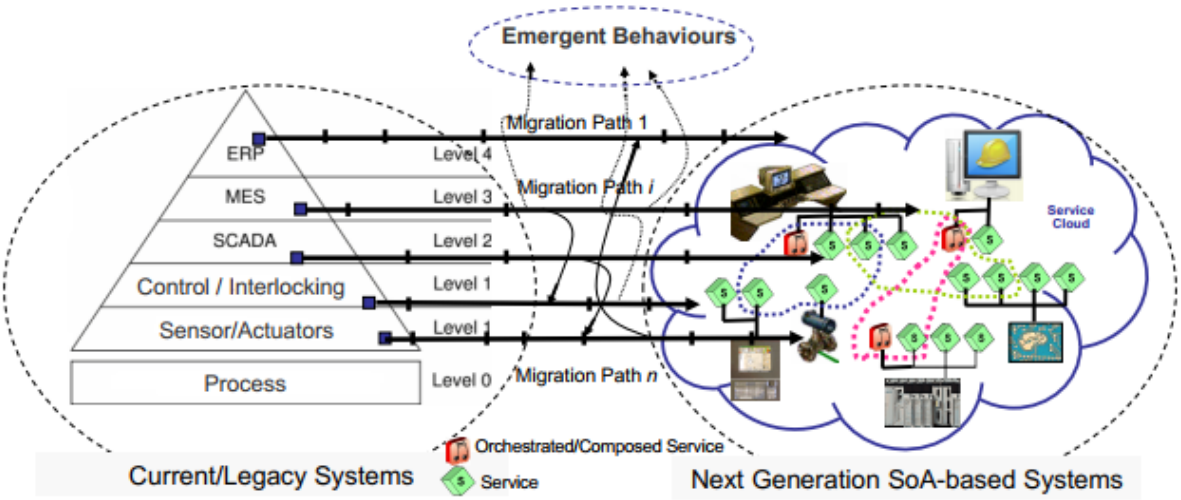
Nikoladis et al. (2015) propõem a interação entre dois conceitos: a utilização de controladores em nuvem com dispositivos baseados em IIoT. Chen, Du e García-Acosta (2010) apresentam a ideia de *Robot as a Service* ou RaaS, sendo esse um serviço na nuvem para acesso a *hardware* e *software* de um robô. Wu et al. (2013) exploram o conceito de *Cloud Manufacturing*, que é um modelo de produção baseado em nuvem. Por fim, Jammes et al. (2014) descrevem o desenvolvimento de serviços para o monitoramento e controle com uso do protocolo como CoAP para acesso a sensores e atuadores.

Um paradigma recente para a indústria tem sido o desenvolvimento de soluções colaborativas através do uso e compartilhamento de serviços para obtenção de uma arquitetura interoperável, distribuída e totalmente integrada através de redes industriais. Nesse sentido, o desenvolvimento de arquiteturas orientadas a serviços (SOA – *Service Oriented Architecture*) para a aplicação da IIoT e I4.0 tem se destacado conforme descrito em Jammes et al. (2014). Colombo, Karnouskos e Bangemann, (2014) discutem o conceito de uma arquitetura de automação baseada no uso da Computação em Nuvem e de Serviços *Web* (*Web Services*) para

comunicação e realização das tarefas entre os diferentes componentes e equipamentos de um sistema industrial. Os autores atestam que SOA é considerada um caminho promissor para concepção do modelo de fábrica do futuro. Um dos principais desafios relacionados à evolução dos sistemas industriais na I4.0 está relacionado com o grau necessário de descentralização e a independência dos sistemas. Nesse contexto, SOA apresenta um grande potencial, a partir da disponibilização de serviços em nuvem alocados em diferentes dispositivos, *gateways* ou sistemas, os quais facilitam e padronizam as interações entre eles.

Colombo et al. (2017) afirmam que a SOA conduz à ideia de que cada *hardware* programável pode ser considerado como um potencial dispositivo que pode ser publicado na forma de serviço virtualizado, o qual permite a sua programação e/ou configuração por meio de aplicativos. Nesse contexto, as funcionalidades dos dispositivos podem ser acessadas por diferentes módulos e sistemas, independentes da heterogeneidade do *hardware/software*, permitindo a interoperabilidade dos sistemas como mostrado da Figura 4. Nessa figura é possível notar que há uma migração da arquitetura legada tradicional hierárquica baseada em camadas definida pela ISA-95 para uma arquitetura SOA em nuvem, onde todos os recursos, equipamentos e sistemas são disponibilizados numa mesma infraestrutura em nuvem de forma padronizada e interoperável.

Figura 4 – Arquitetura SOA Aplicada na Automação de um Processo.



Fonte: Colombo et al. (2017).

1.2 Objetivo

Este trabalho objetiva o desenvolvimento e avaliação de uma nova abordagem para o controle de processos com foco na Indústria 4.0. Nesta abordagem, a implementação das malhas

de controle e automação de processos industriais é realizada através da tecnologia de serviços e baseada no uso do *framework* Moleculer de microserviços. Para isso, o trabalho apresenta o desenvolvimento dos serviços de Aquisição de Dados (DAQ), de Controle PIDPlus e de Controlador Lógico Programável (CLP), bem como a composição desses serviços por orquestração para criação de aplicações de Controle e Supervisão como um serviço.

1.3 Estrutura do Trabalho

Este trabalho de mestrado possui mais sete capítulos, além deste capitulo introdutório. No Capítulo 2 é apresentada uma revisão bibliográfica sobre conceitos e tecnologias relacionadas às aplicações da indústria 4.0. São apresentados também os conceitos das arquiteturas SOA e MOA e aplicações industriais utilizando serviços.

O Capítulo 3 apresenta o *framework* Moleculer, utilizado para a criação de microserviços na plataforma Node.js.

No Capítulo 4 é descrita a proposta desse trabalho de mestrado, que envolve o desenvolvimento de um microserviço DAQ utilizado em conjunto com um microserviço PIDPlus, e também a proposta de desenvolvimento de um microserviço CLP associado ao projeto OPEN PLC.

O Capítulo 5 contém o desenvolvimento da proposta desse trabalho, bem como a apresentação dos resultados obtidos através de experimentos práticos.

As conclusões são apresentadas no Capítulo 6 e em sequência estão as referências bibliográficas e apêndices do trabalho com informações complementares sobre seu desenvolvimento.

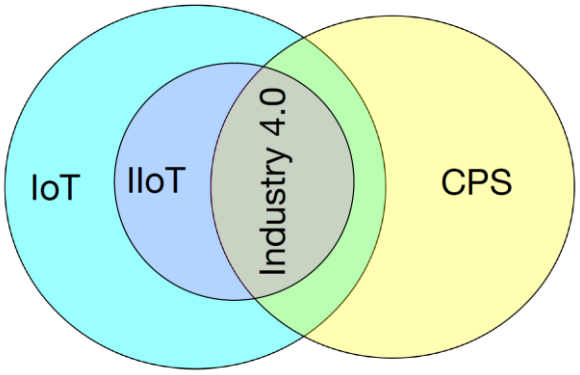
2 REVISÃO BIBLIOGRÁFICA

2.1 Indústria 4.0

O conceito de Indústria 4.0 (I4.0) surgiu na Alemanha em 2011 (JAZDI, 2014) e ganhou visibilidade global, sendo atualmente utilizado para abordar o uso de tecnologias de Internet para elevar a eficiência de produção através de serviços inteligentes (*smart services*) em fábricas inteligentes (*smart factories*). A I4.0 é um novo conceito que representa uma evolução dos sistemas produtivos atuais a partir da convergência entre novas tecnologias de automação (TA) e tecnologia da informação (TI) (BLANCHET et al., 2014).

Do ponto de vista de aplicação, é uma mistura da ideia de IoT com a ideia de Sistemas *Cyber*-Físicos (SISINNI et al., 2018). Sistemas *Cyber*-Físicos (CPS - *Cyber-Physical Systems*) são aqueles nos quais há integração da computação com os processos físicos, ou seja, uma "intersecção" entre o físico e o virtual. De forma geral, são sistemas automatizados que permitem conexão das operações da realidade física com a infraestrutura de computação e comunicação (JAZDI, 2014). A I4.0 representa a interseção entre o sistema CPS com a IoT e sua aplicação industrial (IIoT – *Industrial Internet of Things*). A Figura 5 apresenta as interseções de IoT, CPS, IIoT e Industria 4.0 (SISINNI et al., 2018).

Figura 5 – Representação das intersecções entre IoT, IIoT, CPS e Indústria 4.0.



Fonte: Sisini et al. (2018).

Alguns princípios fundamentais são recorrentes nas diversas definições de Indústria 4.0 introduzidas ao longo do tempo (BASSI, 2017). São eles:

- ✓ Uso extensivo da Internet: o uso de serviços de comunicação cabeada ou sem fio nos dispositivos inteligentes possibilita acesso direto às camadas superiores de processos e serviços. Isto eleva o valor agregado, promove suporte para o uso otimizado de recursos e o controle inteligente do processo de fabricação (JAZDI, 2014). Uma das

principais vertentes é o uso de *big data* para a predição de falhas, de forma que se faça a manutenção antes que ocorra a quebra do equipamento. Isto reduz drasticamente o tempo de parada da planta e os prejuízos decorrentes da interrupção da linha (BASSI, 2017);

- ✓ Flexibilidade: a capacidade de operar linhas produtivas flexíveis, com capacidade de customização e minimização do tempo de reprogramação (*setup*), de forma a atender as necessidades individuais do consumidor sem custo adicional por parada e reprogramação da linha de produção (JAZDI, 2014). Algumas tecnologias habilitadoras típicas são a impressão 3D e a rastreabilidade e identificação do produto na linha de produção (BASSI, 2017);
- ✓ Comunicação, Virtualização e CPS: a criação de modelos unificados em contraposição aos diversos protocolos industriais atuais (AS-i, PROFIBUS, PROFINET, CAN, etc.) que em geral são incompatíveis entre si ou de difícil interoperabilidade. A disponibilização de um *framework* comum de comunicação permite o uso de virtualização, conectando os sistemas físicos a contrapartes virtuais (BASSI, 2017).

Segundo Kagermann, Wahlster e Helbig (2013), o potencial da I4.0 fica evidente devido à algumas características e possibilidades, como:

- ✓ Atender aos requisitos individuais do cliente: A linha de produção pode se modificar para atender critérios individuais, específicos do cliente. Mudanças de última hora e volumes de produção muito baixos (tamanho de lote de 1), também são possíveis;
- ✓ Flexibilidade: É possível configurar dinamicamente diferentes aspectos dos processos de negócios, tais como qualidade, tempo, risco e preço. Isso possibilita a otimização do uso de materiais, os processos de engenharia podem ser tornados mais ágeis, processos de fabricação podem ser alterados, a escassez temporária de algum material (devido a problemas de fornecimento, por exemplo) pode ser compensada e grandes aumentos no tamanho dos lotes produzidos podem ser alcançados em um curto espaço de tempo;
- ✓ Tomada de decisão otimizada: A I4.0 permite a verificação antecipada das decisões de projeto, respostas mais flexíveis à interrupção e otimização global em todos os setores de uma empresa;
- ✓ Produtividade e eficiência dos recursos: A mesma motivação que resultou nas revoluções industriais anteriores impulsiona a I4.0, a obtenção da maior produção possível a partir de um determinado volume de recursos (produtividade dos recursos) e utilização da menor quantidade possível de recursos para produzir uma determinada

> quantidade de produtos (eficiência de recursos). Os processos de fabricação podem ser otimizados para cada caso individual, até mesmo sem parar a produção, sendo continuamente otimizados em termos de consumo de recursos e energia ou de redução de suas emissões.

As aplicações da I4.0 são baseadas na ideia de que onde tudo estará conectado para que as melhores decisões de produção, custo e segurança sejam tomadas, tudo sob demanda e em tempo real. Para que isso aconteça, é necessário a interconexão de dados de automação e infraestrutura de redes de TI de forma segura e confiável (WOLLSCHLAEGER; SAUTER; JASPERNEITE, 2017). As arquiteturas industriais de automação e controle tradicionais não fornecem, principalmente, estes requisitos de interoperabilidade e integração vertical e horizontal de equipamentos e sistemas (BORANGIU et al., 2019; SISINNI et al., 2018). Em consequência, torna-se necessário o desenvolvimento de novas arquiteturas industriais que forneceram tais requisitos e suportem as demandas tecnologias relacionadas à I4.0. Nesse sentido a utilização da Arquitetura Orientada a Serviços (SOA) surge com uma solução para a integração das tecnologias já existentes para a I4.0.

2.2 Arquitetura Orientada a Serviços (SOA)

A Arquitetura Orientada a Serviços (SOA - *Service Oriented Architecture*) é uma arquitetura de sistemas de Tecnologia de Informação (TI) na qual se busca fragmentar as aplicações convencionais em “Serviços”, que são funções e processos individuais de negócio (XIAO; WIJEGUNARATNE; QIANG, 2017). Na prática, trata-se de uma arquitetura modular, na qual os elementos devem ser compostos e decompostos de acordo com as diferentes necessidades do negócio. Neste tipo de sistema, cada serviço deve ser passível de descoberta e independente quanto à plataforma e à linguagem (THEORIN; HAGSUND; JOHNSSON, 2014), com o objetivo de obter baixo acoplamento de um serviço em relação ao outro, escalabilidade e a capacidade de manter uma arquitetura distribuída.

Do ponto de vista tecnológico, SOA é uma nova filosofia de projeto caracterizada pela modularidade, separação de tarefas, reuso de serviços e composição. Pode ser vista também como uma nova metodologia de programação baseada em padrões e ferramentas que envolvem o uso de serviços *web* em larga escala (ORDANINI; PASINI, 2008).

No contexto computacional, os serviços são módulos de negócio que possuem interfaces de conexão que são invocadas via mensagens (DUSTDAR; PAPAZOGLOU, 2008). Essas interfaces disponibilizam recursos sem que a implementação dos serviços seja conhecida, sendo

esses serviços processos independentes que são descritos, disponibilizados, localizados e invocados por meio de redes de comunicação. Os serviços podem interagir através de redes de comunicação com simples respostas a requisições de processos até a execução de ações e tarefas complexas que exigem uma comunicação entre provedores e consumidores de serviços virtuais.

Um grande número de pesquisas têm focado no desenvolvimento de arquiteturas orientadas a serviços (SOA) com o objetivo de oferecer uma arquitetura para suportar a propagação e a utilização de serviços virtuais reutilizáveis (XIAO; WIJEGUNARATNE; QIANG, 2017; JAMMES et al., 2014). Para Sheng et al., (2014), a SOA é uma arquitetura para organizar infraestruturas e aplicações em um conjunto de serviços passíveis de interação, onde as informações e os recursos ficam disponíveis para todos os clientes como serviços independentes que são acessados de um modo padrão.

Nas pesquisas de Souit et al. (2013) e Sheng et al. (2014), a SOA estabelece uma arquitetura que permite que os serviços sejam publicados, descobertos e consumidos por aplicações ou outros serviços. Para Xiao, Wijegunaratne e Qiang, (2017), a SOA é a integração e a colaboração de recursos distribuídos com o conceito de serviço. Na SOA os serviços podem ser chamados de maneira independente por qualquer cliente (externos ou internos) para processar funções de forma coordenada para desenvolver novas funcionalidades para os processos existentes.

Os serviços podem ser implementados considerando a chamada de outros serviços existentes. Esta abordagem é conhecida como composição de serviços, apontada como um dos principais aspectos da computação orientada a serviço que contribui para o reuso dos serviços. Segundo Dustdar e Papazoglou (2008) as técnicas mais utilizadas para composição de serviços são a orquestração e a coreografia.

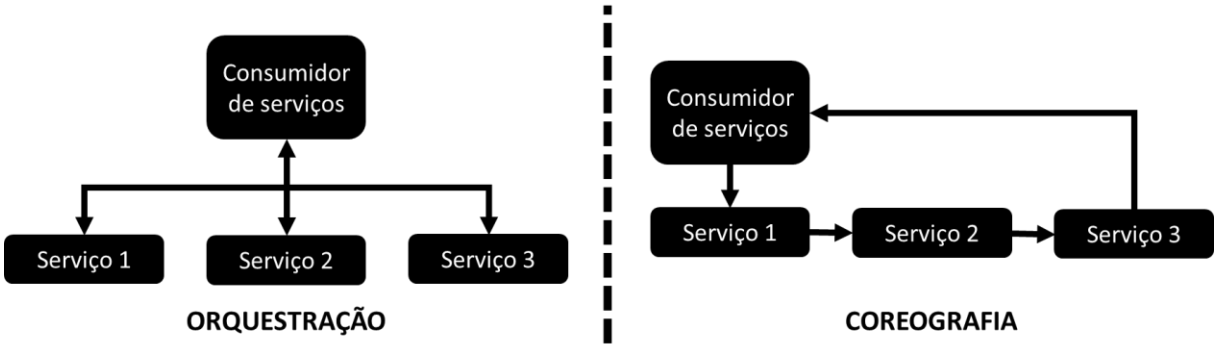
2.2.1 Composição de Serviços

Quando se discutem arquiteturas baseadas em serviços, um ponto que deve ser levado em conta é o tipo de organização que se deve adotar para que todo o conjunto funcione de forma satisfatória. Richards (2015) apresenta dois conceitos de composição de serviços: a orquestração e a coreografia. A Figura 6 apresenta um esquema comparativo entre orquestração e coreografia de serviços.

A orquestração representa a composição de serviços para criar um novo serviço ou para resolver uma tarefa de um processo de negócio. Neste caso, existe sempre um processo central (mestre), sendo um serviço ou uma atividade de negócio que coordena a chamada de outros serviços para compor uma função de maior relevância. Para este tipo de composição de serviço,

cada serviço participante não tem conhecimento de que faz parte de uma composição de serviço de maior hierarquia, com exceção do processo mestre. Somente o processo mestre detém a inteligência sobre a aplicação completa, em outras palavras, a orquestração de serviços se refere à coordenação centralizada de diversos serviços através de um mediador, sendo que este mediador tem a responsabilidade de coordenar a execução dos serviços, de forma que seja obedecida uma ordem predefinida para a execução.

Figura 6 – Comparativo entre orquestração e coreografia de serviços.



Fonte: Adaptado de Richards (2015).

Na coreografia, os serviços são pré-determinados antes da sua execução. Quando um serviço é acionado e envia uma mensagem, outros serviços podem estar programados antecipadamente para receber ou não essa mensagem e dispararem outras ações. Na composição de serviços por coreografia, não existe um processo mestre que controla e coordena os demais processos. Nessa configuração, cada processo envolvido tem o conhecimento de que faz parte de uma composição de serviços e que precisa interagir com outros serviços/aplicações de maneira ordenada para que a composição resultante tenha sucesso. Cada serviço sabe quando atuar, com quais outros serviços interagir, quais operações deve executar e quais mensagens deve trocar, sendo a execução, portanto, descentralizada.

A coreografia de serviços também pode ser definida como um conceito que dispensa o elemento central coordenador, partindo do pressuposto que cada serviço sabe qual é a sua parte dentro do contexto e a aplicação final é criada pela soma dos serviços individuais (BUTZIN; GOLATOWSKI; TIMMERMANN, 2016). Neste tipo de implementação, cada serviço é responsável por acionar o seu sucessor, formando o que se chama “cadeia de serviços”, de forma semelhante a uma coreografia de dança, na qual cada integrante conhece a sua função dentro do contexto e a executa sem que haja um comandante central.

Do ponto de vista das arquiteturas SOA, ambos os modelos podem ser implementados. Na prática, porém, predomina o uso de coreografia, na qual há maior flexibilidade na gestão

dos serviços (BUTZIN; GOLATOWSKI; TIMMERMANN, 2016). Quando um novo serviço é adicionado dentro de um contexto de orquestração, a aplicação mediadora deve ser alterada de forma a incluir o novo serviço. Num contexto de coreografia, por sua vez, basta que os serviços relacionados sejam alterados, de forma que o novo serviço seja acionado quando necessário.

Um dos pontos relevantes quanto à escolha do tipo de organização é a necessidade ou não de se verificar que todos os serviços foram executados com sucesso. Em serviços orquestrados, esta verificação é possível diretamente através da instância do mediador. Quando os serviços são coreografados, esta verificação só pode ser implementada através de um serviço de monitoramento dos demais (BUTZIN; GOLATOWSKI; TIMMERMANN, 2016).

2.2.2 Microserviços

Evolução da SOA, os microserviços são um estilo arquitetônico em que as aplicações são decompostas em serviços acoplados, oferecendo modularidade, tornando as aplicações mais fáceis de desenvolver, testar, implantar e, o mais importante, alterar e manter. Microserviços são serviços com poucas responsabilidades, pequenos e autônomos que podem trabalhar de forma independente ou em conjunto com outros serviços. O conceito foi definido em 2014 e, resumidamente, descreve uma interpretação mais concreta da SOA (BUTZIN; GOLATOWSKI; TIMMERMANN, 2016).

Cada microserviço é definido como uma entidade separada e autônoma, que pode ser implementada em diferentes máquinas e ainda assim trabalhar de forma coletiva, possuindo a capacidade de executar seus processos independentemente caso haja falha de algum outro serviço (MOREIRA; BEDER, 2016). Os microserviços são um padrão arquitetural que não foi inventado, mas que surgiu de boas práticas de desenvolvimento de *software* com a intenção de criar sistemas altamente escaláveis e rápidas mudanças dos modelos de negócio na nuvem. Alguns exemplos de sucesso são as empresas Netflix, Amazon e Soundcloud (BUTZIN; GOLATOWSKI; TIMMERMANN, 2016).

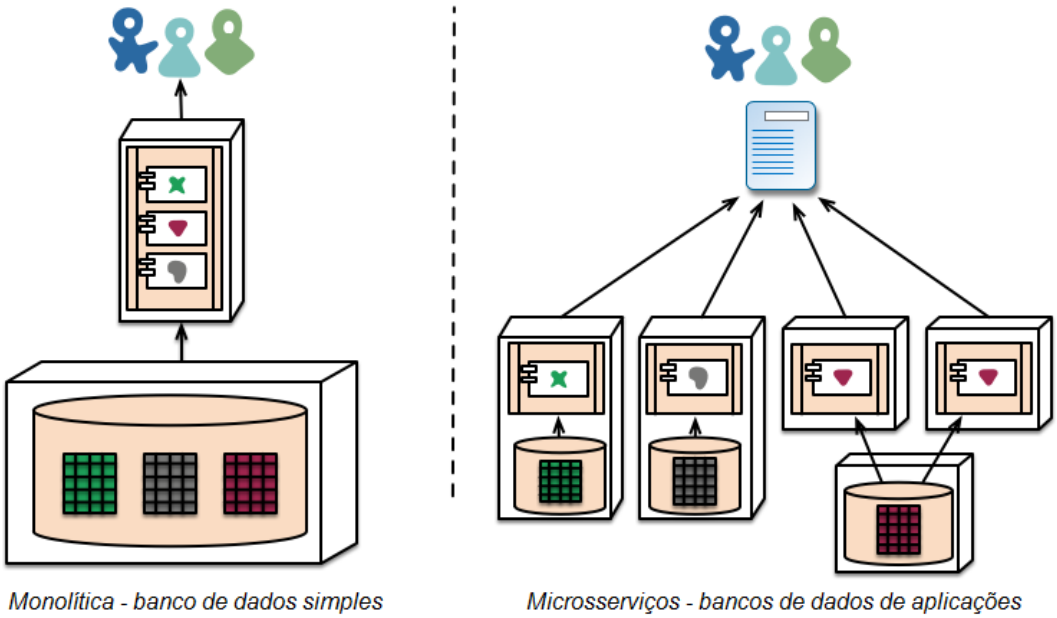
Lewis e Fowler (2014) descrevem todas as características que definem a estrutura de um microserviço. Xiao, Wijegunaratne e Qiang, (2017) descrevem um microserviço como um aplicativo flexível, escalável, interoperável, distribuído e totalmente integrado através de redes. As principais características que definem um microserviço são (LEWIS; FOWLER, 2014; NEWMAN, 2015):

- ✓ Cada microserviço implementa um único recurso de negócios ou funcionalidade;

- ✓ Um microserviço é o suficiente simples para ser desenvolvido e mantido por um único programador;
- ✓ Microserviços são executados em processos separados, comunicando-se por meio de padrões de mensagens ou APIs bem definidas;
- ✓ Microserviços não compartilham armazenamentos de dados. Cada microserviço é responsável por gerenciar seus próprios dados;
- ✓ Microserviços têm bases de código separadas e não compartilham código-fonte. No entanto, eles podem usar bibliotecas de utilitários comuns;
- ✓ Cada microserviço pode ser implantado e atualizado de forma independente de outros serviços.

Ao ser agnóstico de dispositivos e plataformas, os microserviços fornecem flexibilidade para o sistema desenvolvido. Os microserviços têm diferentes formas de comunicação e processamento sendo mais um dos atributos que os distinguem do SOA tradicional. A Figura 7 apresenta um esquemático dos microserviços.

Figura 7 – Esquemático da composição dos microserviços.

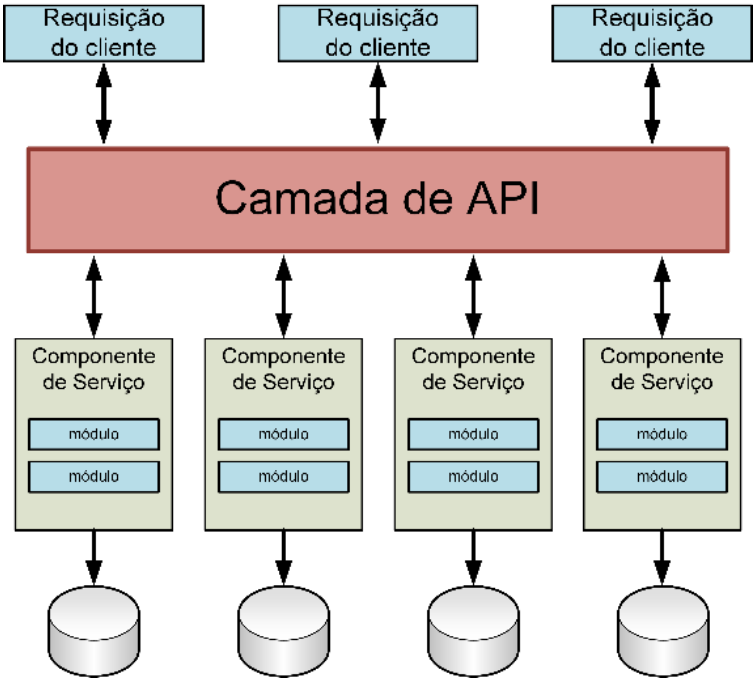


Fonte: Adaptado de Lewis e Fowler (2019).

A Figura 8 apresenta a arquitetura típica de microserviços. Nota-se que há apenas dois componentes principais: os componentes de serviço (microserviços em si) e uma camada de interface de programação de aplicativos (API – *Application Programming Interface*). Do ponto de vista de implementação, pode haver alguns outros componentes, tais como o serviço de

registro e descoberta, monitoração e outros, entretanto do ponto de vista arquitetural, tais componentes seriam considerados serviços de infraestrutura (RICHARDS, 2015).

Figura 8 – Arquitetura típica de microserviços.



Fonte: Adaptado de Richards (2015).

2.3 Aplicações Industriais usando Serviços

As Tecnologias de Informação e Comunicação (TIC) estão cada vez mais integradas aos sistemas industriais. Isso leva a uma dependência significativa de Tecnologia da Informação (TI), à medida que nos aproximamos da visão da IIoT, onde inúmeros dispositivos interconectados (sensores, atuadores, SCADA, DCS e dispositivos móveis) fornecerão e consumirão informações disponíveis nas redes de comunicação. A Arquitetura Orientada a Serviços (SOA) é uma solução para realizar a interação entre camadas de dispositivos e sistemas heterogêneos e garantir níveis mais altos de interoperabilidade (DELSING et al., 2011).

O conceito de serviços permite que os dispositivos, equipamentos e sistemas industriais passem a ser clientes ou servidores de dados, a depender da necessidade do processo ou aplicação, e que haja interação entre eles, de forma transparente, não importando os níveis hierárquicos do processo industrial a que eles pertençam (MORAES, 2017). Além disso, a arquitetura SOA permite que qualquer aplicação industrial, no contexto de CPS, IIoT e I4.0, possam ser rapidamente compostas pela seleção e combinação de novos serviços e funcionalidades disponibilizadas como um serviço em nuvem.

Nos últimos anos, diversos trabalhos desenvolveram arquiteturas SOA e avaliaram sua a utilização em aplicações industriais no contexto de CPS, IIoT e I4.0. Leitão et al. (2016) apresentam uma revisão de algumas iniciativas financiadas pela União Europeia, que são os projetos SOCRADES, IMC-AESOP, GRACE e ARUM. No trabalho são apresentadas as arquiteturas SOA desenvolvidas e discutidas implementações e casos de estudo onde estas arquiteturas foram testadas e validadas, com destaque ao projeto IMC-AESOP, que é bastante relacionado à proposta deste trabalho. O projeto IMC-AESOP (*ArchitecturE for Service-Oriented Process*) (IMC-AESOP, 2011) desenvolveu uma arquitetura SOA em nuvem para aplicações industriais focada principalmente em aplicações de controle de processos, sistemas SCADA (*Supervisory Control and Data Acquisition*) e DCS (*Distributed Control Systems*). O trabalho de Karnouskos et al. (2014) descreve a concepção dos serviços e o mecanismo de composição de serviços para obtenção de outras funcionalidades, destacando a comunicação via *Web Service* entre os serviços através do Perfil de Dispositivos para Serviços da *Web* (DPWS).

Ollinger et al. (2014) sugerem um conceito de SOA-PLC que permite uma geração totalmente automatizada e a implantação de funções de dispositivo de campo como serviços implementados com o DPWS. Além disso, o conceito é demonstrado em um estudo de caso usando um CLP convencional da Phoenix Contact e a ferramenta de orquestração DPWS *JGrafchart*, que supera as limitações atuais dos sistemas CLP existentes, atendendo às demandas de um ambiente de produção em rápida mutação e seus requisitos desafiadores de flexibilidade e reusabilidade.

Nesse conceito, os blocos de funções são agrupados e encapsulados e, no campo de serviços da *Web*, as funções do dispositivo são disponibilizadas fora do PLC em um nível mais alto da pirâmide de automação. Consequentemente, a lógica de controle de um SOA-PLC não é mais realizada por um código sequencial inflexível, mas por uma orquestração dinâmica de serviços. A orquestração pode ser feita por qualquer ferramenta de orquestração DPWS como o *JGrafchart* em qualquer PC convencional ou até mesmo em outro CLP.

Espí-Beltran, Gilart-Iglesias & Ruiz-Fernandez (2017) utilizam o padrão REST para desenvolvimento de uma arquitetura SOA industrial. A arquitetura no seu nível inferior permite a comunicação com dispositivos e equipamentos legados através de protocolos de comunicação tradicionais, como o Modbus TCP/IP ou através de conexão direta via GPIO. No nível mais alto da arquitetura, serviços são disponibilizados através de comunicação via protocolos compatíveis com o padrão REST. Os resultados experimentais deste trabalho demonstram que o desempenho da arquitetura é adequado em face dos requisitos de comunicação (tempo de resposta, de transmissão e *jitter*) industriais atuais.

Carlsson et al. (2018) discutem sobre os desafios da migração de sistemas de controle de processo tradicionais (legados) para SOA, exemplificando através de um caso de estudo de migração de uma planta de grande porte com diferentes equipamentos e redes de comunicação. Neste trabalho foi realizada a troca dos sistemas de controle baseados em Controlador Lógico Programável (CLP) por SOA. Um ponto importante realçado, para manter a compatibilidade da SOA com as soluções de rede do sistema legado, foi o desenvolvimento de um serviço de mediação. Esse serviço faz a conexão/tradução entre as redes instaladas (Profibus-DP, Modbus TCP/IP e OPC UA) e o padrão de comunicação baseado em serviços *Web* da SOA.

Bigheti et al. (2018) apresentam uma arquitetura orientada a microserviços (MOA) em nuvem para aplicações de IIoT e I4.0, discutindo as características, detalhes operacionais, vantagens e potencias de aplicação. Os autores explicam que a integração de diferentes tecnologias usando uma arquitetura SOA através do uso e compartilhamento de microserviços fornece interoperabilidade para as aplicações. Por fim, demonstram que os testes operacionais iniciais da arquitetura realizados com a implementação de uma malha de controle e supervisão de uma variável de processo usando microserviços validam a proposta.

Ainda que uma grande quantidade de trabalhos relacionados ao desenvolvimento e implementação de SOA para aplicações industriais possa ser encontrado na literatura, o mesmo não acontece quando se foca em arquiteturas orientadas a microserviços (MOA). Francesco, Malavolta & Lago (2017) apresentam uma revisão sobre a adoção industrial de MOA e concluem afirmando que o uso de MOA ainda é bastante incipiente. A maioria dos trabalhos existentes focam em proposições de arquiteturas, discussão de vantagens e desvantagens de uso e análise dos desafios para migração e implantação industrial de MOA (FRANCESCO, LAGO & MALAVOLTA, 2018; SARKAR, VASHI & ABDULLA, 2018).

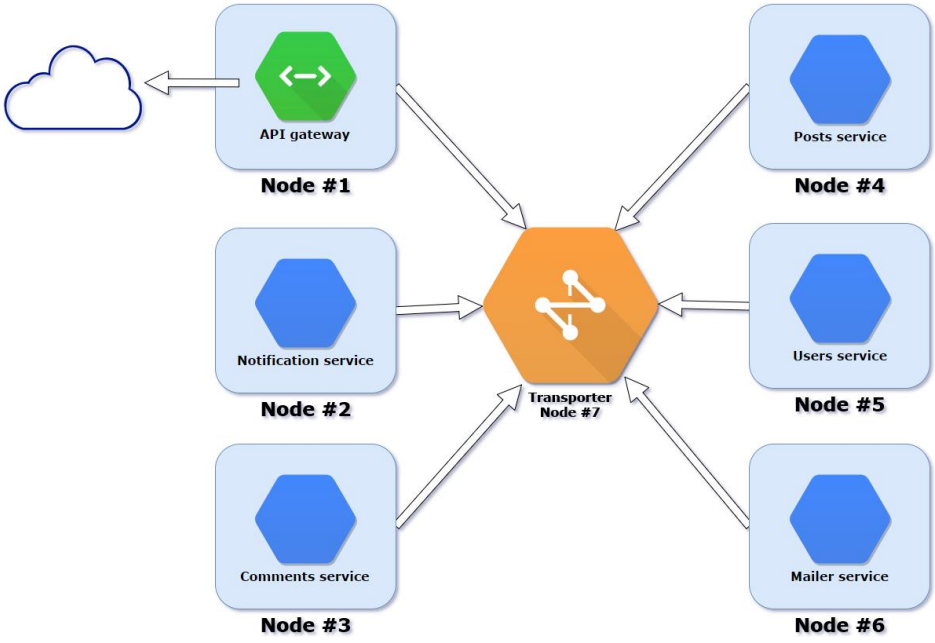
Os levantamentos realizados demonstram a importância da pesquisa e desenvolvimento de SOA, e mais recentemente MOA, para fornecer os novos requisitos das aplicações de controle e automação de processos industriais no contexto da IIoT e I4.0. Mais importante ainda, devido à carência atual, é a discussão de resultados experimentais obtidos com o desenvolvimento e implementação dessas arquiteturas, os quais permitam avaliar questões práticas relacionados ao desempenho, vantagens e dificuldades deste tipo de aplicação.

3 *FRAMEWORK* MOLECULER

Um *framework* representa uma estrutura base ou uma plataforma de desenvolvimento, que contém ferramentas, bibliotecas, sistemas e componentes que simplificam o processo de desenvolvimento de uma solução específica. O Moleculer é um *framework* de código aberto utilizado na criação de microserviços em linguagem JavaScript para a plataforma Node.js. Ele ajuda a construir serviços eficientes, confiáveis e escaláveis, e fornece muitos recursos para construir e gerenciar microserviços (MOLECULER, 2018).

A Figura 9 apresenta a arquitetura orientada a microserviços (MOA) do *framework* Moleculer, na qual os serviços são executados em nós individuais que se comunicam via protocolo de comunicação (*Transporter*) através de um *broker*, ou seja, um intermediário de envio de mensagens. Nesta arquitetura, a latência da comunicação entre os serviços não é insignificante, mas os serviços podem ser replicados para proporcionar resiliência e tolerância a falhas. Outras estruturas de interconexão entre os serviços também são permitidas no Moleculer, permitindo a melhor configuração para cada aplicação.

Figura 9 – Arquitetura Orientada a Microserviços do Moleculer.



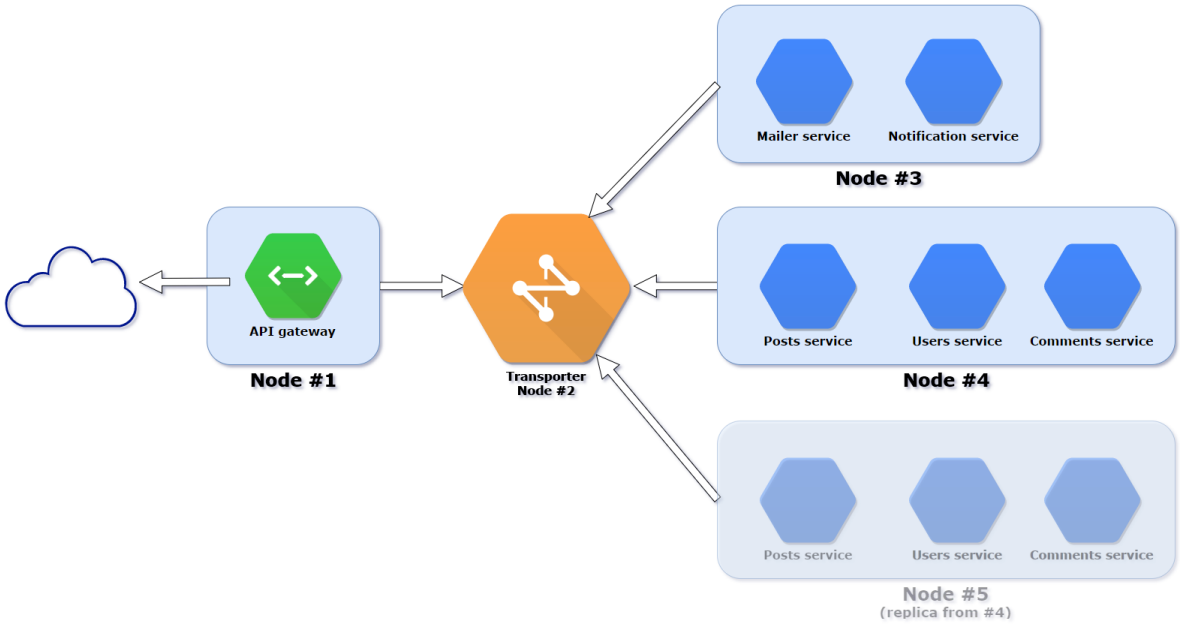
Fonte: Moleculer (2018)

A replicação de serviços consiste na implementação de dois (ou mais) serviços idênticos e com o mesmo nome em instâncias de execução diferentes. Na prática, o Moleculer pode direcionar as requisições para qualquer um dos serviços replicados, dependendo da disponibilidade de cada um dos serviços. Existe também a possibilidade de reuso dos códigos,

porém estabelecendo cada serviço com um nome distinto, sendo que nesse caso, o Moleculer direciona as requisições para cada serviço especificamente. Para melhor entendimento do reuso do código, se existe um serviço com a funcionalidade de controlar o nível de algum liquido em um tanque, e surge a necessidade de se implementar esse mesmo controle para outros tanques, por exemplo, o reuso do código pode ser aplicado, tendo as mesmas funcionalidades para todos os controles, porem de maneira individual.

A Figura 10 apresenta um exemplo da arquitetura MOA no qual é possível visualizar o agrupamento de vários serviços em um mesmo nó. Os serviços que estão alocados em um mesmo Nó não necessitam se comunicar por meio do *Transporter*, o que reduz a latência da comunicação. Caso um nó ou serviço esteja sobrecarregado, é possível realizar a replicação deste, conforme mostrado no exemplo onde o Nó 4 (*Node* #4) é replicado no Nó 5 (*Node* #5).

Figura 10 – Exemplo de Interconexão de Serviços do Moleculer.



Fonte: Moleculer (2018).

Na arquitetura do Moleculer não existe qualquer tipo de hierarquia ou prioridade entre os microserviços que compõem. Uma grande vantagem desse *framework* é que todos os microserviços desenvolvidos possuem disponível um recurso automático de registro e descoberta, sendo assim, todos os serviços existentes são informados após a criação de um novo serviço ou após a disponibilização de uma nova funcionalidade em um serviço. Outro recurso importante é o balanceamento automático de carga, o qual tem função de distribuir dinamicamente a carga da comunicação entre os microserviços uniformemente.

Os quadrados de cor azul na arquitetura (Figura 9 e Figura 10) representam os nós de gerenciamento do Moleculer para suporte ao desenvolvimento, descoberta e configuração dos (micro) serviços. Cada Nó é responsável pelo gerenciamento dos seus dados, possuindo seu próprio banco de dados, e pode conter um ou mais serviços. Os serviços podem oferecer e executar diferentes tarefas que são chamadas de ações (exemplo: um serviço de aquisição de dados pode oferecer uma ação de aquisição de dados de entrada e uma ação de atualização de dados de saída). Na Figura 9 e na Figura 10 cada serviço disponibilizado na arquitetura é representado com um hexágono azul.

O hexágono verde representa o (micro) serviço de *gateway* (API *Gateway*), o qual tem a função de interface de conexão dos serviços internos (hexágonos azuis) com aplicações externas através de chamadas via rede ou nuvem por meio da arquitetura de *software* REST. A comunicação entre os microserviços é realizada via um serviço de mensagens (*Transporter*) também conhecido como *Broker* de mensagens, representado pelo hexágono laranja.

O serviço de mensagens (*Transporter*) é um dos componentes mais importantes da arquitetura, pois é o mecanismo que coordena o envio e a recepção de mensagens em uma fila. O *Transporter* é capaz de enfileirar e manter as mensagens até serem processadas por consumidores. O produtor simplesmente envia a mensagem para o *Transporter*, sem a necessidade de um mecanismo de descoberta de serviços.

A comunicação baseada em mensagens também suporta uma variedade de padrões de comunicação, incluindo os pedidos de caminho único (*one-way requests*) e *publish-subscribe*. Uma vantagem desse serviço no Moleculer é que se um mesmo serviço é disponibilizado em múltiplas instâncias em diferentes nós (maior disponibilidade), as requisições recebidas pelo *Transporter* serão automaticamente balanceadas entre os nós que estão online naquele momento e que podem executar o serviço requisitado. Além disso, com o uso do *Transporter* a mudança de um mecanismo de comunicação para outro (entre os diferentes disponíveis no Moleculer) é transparente.

No Moleculer, o serviço *Transporter* utiliza um protocolo de comunicação para se comunicar com os outros microserviços. O grande benefício dessa abordagem é o desacoplamento entre produtores e consumidores de dados e eventos. O desacoplamento simplifica o desenvolvimento e aumenta a disponibilidade, em comparação ao uso de transações distribuídas. Caso nenhum consumidor esteja disponível para processar um evento, o *Transporter* irá enfileirar o evento até que a mensagem possa ser processada.

O serviço *Transporter* visa estabelecer a comunicação entre sistemas completamente diferentes de uma maneira confiável. O *Broker* de mensagem faz com que seja possível integrar tais sistemas de maneira que eles possam trocar dados de forma desacoplada. A troca de

informações entre os microserviços utilizando o *Transporter* é realizado pelo módulo *Serializers*, responsável pela serialização dos dados das mensagens.

Os principais recursos para microserviços que compõem o *framework* Moleculer são (MOLECULER, 2018):

- ✓ Conceito de requisição-resposta;
- ✓ Arquitetura baseada em eventos;
- ✓ Suporte a *middlewares* para Node.js;
- ✓ Suporte ao armazenamento em *cache* de variável;
- ✓ Diversas opções de comunicação (*Transporters);*
- ✓ Diversas opções de *serializers*: JSON, MsgPack, *Protocol Buffer* e etc;
- ✓ Suporte ao desenvolvimento de múltiplos serviços em um mesmo nó;
- ✓ Suporte nativo para o registro de serviços;
- ✓ Descoberta automática de serviços;
- ✓ API para interface com aplicações externas.

O *framework* Moleculer proporciona o desenvolvimento de um estilo arquitetônico em que as aplicações são decompostas em serviços de baixo acoplamento, oferecendo modularidade e tornando mais simples a sua implementação. As principais classes de componentes operacionais do *framework* Moleculer são:

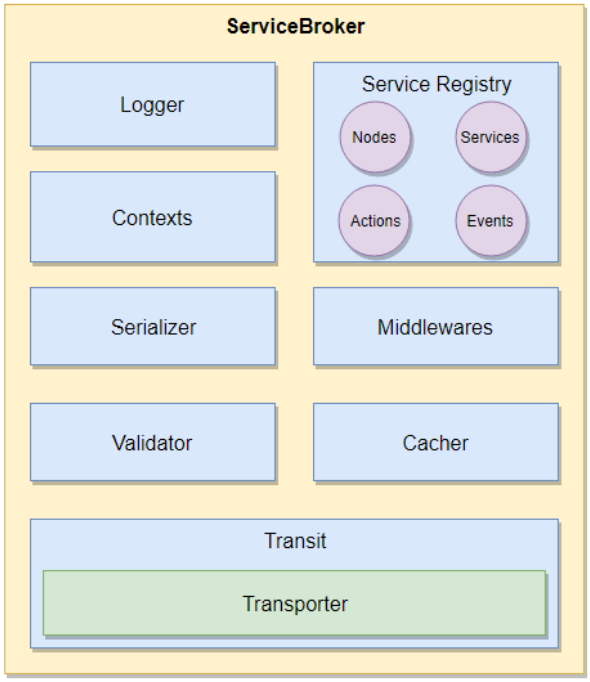
- ✓ *ServiceBroker*;
- ✓ *Service*.

3.1 *Service Broker*

O *ServiceBroker* é o principal componente da estrutura do Moleculer, pois ele engloba as características do *framework* como mostra a Figura 11. Ele lida com serviços, chama ações, emite eventos e se comunica com nós remotos. Também é responsável pela configuração dos nós como: nome do nó, registro de serviços, descoberta automática de serviços, *cache* de variáveis, *serializer*, *middleware* e *transporter*.

Para instanciar um *Node* (nó), é necessário carregar o módulo Moleculer no Node.js. Após isso, pode-se criar um novo objeto de *ServiceBroker* e em seguida configurar as operações de funcionamento do microserviço, como pode ser visto na Figura 12 e na Figura 13. É necessário criar uma instância do *ServiceBroker* para cada nó (MOLECULER, 2018).

Figura 11 – Estrutura do *ServiceBroker*.



Fonte: Moleculer (2018).

Figura 12 – Configuração padrão do *ServiceBroker*.

```
const {ServiceBroker} = require ( "moleculer" ); corretor
const = new ServiceBroker ();
```

Fonte: Moleculer (2018).

Figura 13 – Configuração personalizada do *ServiceBroker*.

```
const { ServiceBroker } = require("moleculer");
const broker = new ServiceBroker({
    logLevel: "info"
});
```

Fonte: Moleculer (2018).

3.2 Serviço (*Service*)

O Serviço representa um microserviço no *framework* Moleculer, onde é possível definir ações e disparar eventos. Para criar um serviço, você deve definir um esquema (*schema*) contendo algumas partes principais: nome, versão, configurações, ações, métodos e eventos (MOLECULER, 2018).

Nome (*name*): Nome do microserviço. É a primeira parte do nome da ação quando se chama o microserviço. O *name* é uma propriedade obrigatória na criação do microserviço.

Figura 14 – Modelo de microserviço do Moleculer.

```
const { ServiceBroker } = require("moleculer");

const broker = new ServiceBroker();

broker.createService({
    name: "math",
    actions: {
        add(ctx) {
            return Number(ctx.params.a) + Number(ctx.params.b);
        }
    }
});

broker.start()
    // Call service
    .then(() => broker.call("math.add", { a: 5, b: 3 }))
    .then(res => console.log("5 + 3 =", res))
    .catch(err => console.error(`Error occured! ${err.message}`));
```

Fonte: Moleculer (2018).

Versão (*version*): Essa propriedade é usada para executar várias versões do mesmo microserviço com ações diferentes.

Configurações (*settings*): Essa propriedade tem a função de configuração dos recursos disponíveis nos microserviços. Por exemplo, a configuração da porta de comunicação dos comandos que serão enviados através do microserviço *Gateway*. Essas configurações são enviadas durante o procedimento de descoberta do serviço;

Ações (*actions*): As ações são os métodos que podem ser chamados no serviço. A chamada de uma ação representa uma chamada de procedimento remoto (RPC – *Remote Procedure Call*). Tem parâmetros de solicitação e retorna a resposta, como uma solicitação HTTP (MOLECULER, 2018).

Como exemplo, representado pela Figura 15, quando houver a chamada da ação *"add"* será retornado o valor da soma de 2 números, e quando for solicitada a ação *"sub"* retornará o valor da subtração de 2 números.

A propriedade *Action* pode ter dois tipos de chamadas. As chamadas internas que são executadas pela função *broker.call* ou externas através do microserviço *Gateway*. As chamadas externas das ações utilizam métodos públicos que podem ser chamados por qualquer objeto programado no microserviço.

Figura 15 – Esquema de serviço simples para definir duas ações.

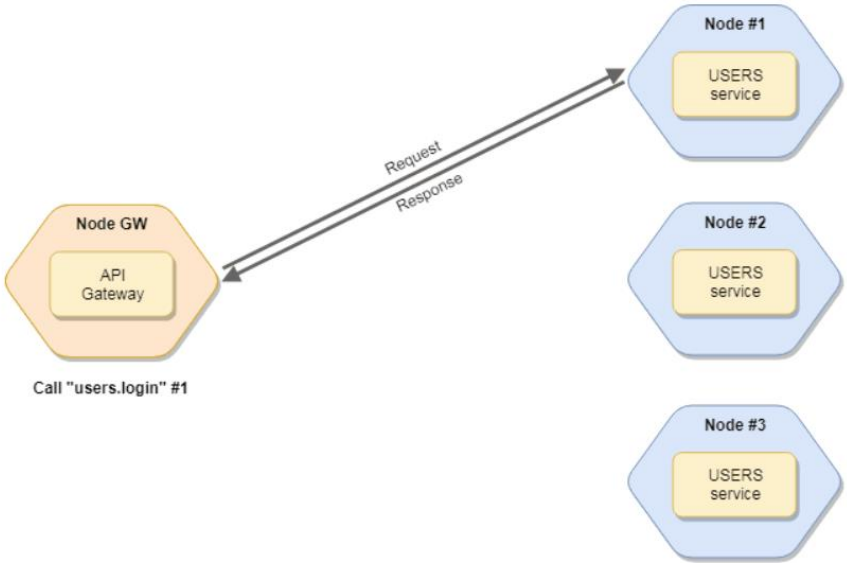
```
{
    name: "math",
    actions: {
        add(ctx) {
            return Number(ctx.params.a) + Number(ctx.params.b);
        },

        sub(ctx) {
            return Number(ctx.params.a) - Number(ctx.params.b);
        }
    }
}
```

Fonte: Moleculer (2018).

A Figura 16 apresenta o método de comunicação por RPC do Moleculer. RPC é uma tecnologia de comunicação entre processos que permite a um programa de computador chamar um procedimento em outro espaço de endereçamento (geralmente em outro computador, conectado por uma rede). O programador não se preocupa com detalhes de implementação dessa interação remota do ponto de vista do código, portanto a chamada se assemelha às de procedimentos locais.

Figura 16 – Método de comunicação RPC das chamadas externas de ações.



Fonte: Moleculer (2018).

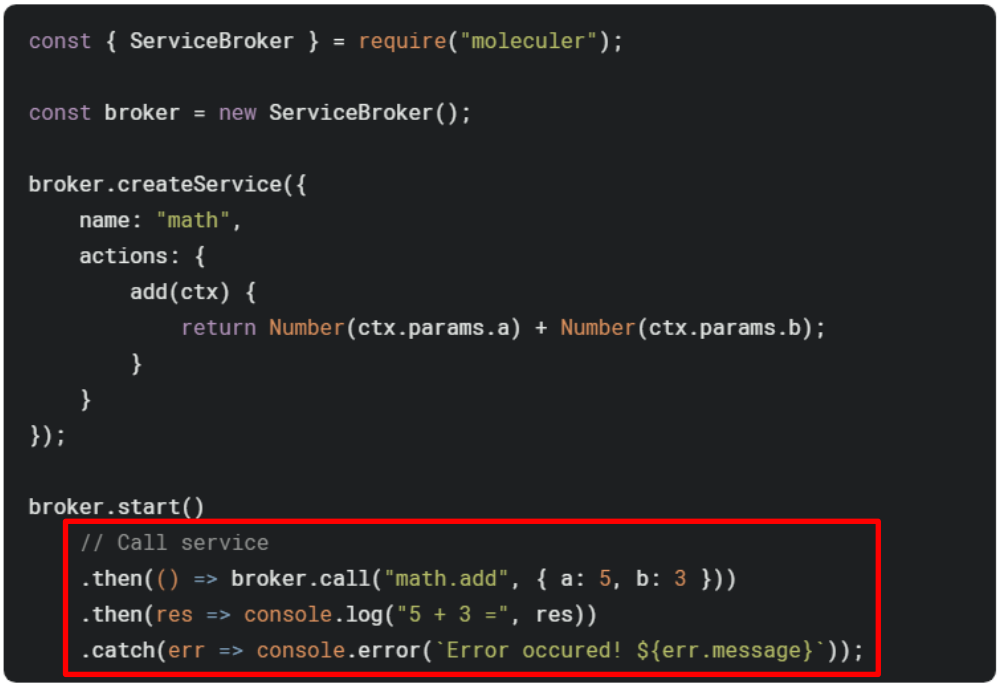
A RPC é uma tecnologia popular para a implementação do modelo cliente-servidor de computação distribuída. Se uma chamada de procedimento remoto for iniciada pelo cliente enviando uma mensagem para um servidor remoto para executar um procedimento específico, uma resposta é retornada ao cliente. Se a aplicação possuir várias instâncias de serviços, o

ServiceBroker fará o balanceamento de carga da solicitação entre as instâncias de maneira uniforme e dinâmica.

O método *broker.call* é utilizado para chamar uma ação interna. Quando o método *broker.call* é acionado, o *ServiceBroker* procura o microserviço em um Nó que tem uma determinada ação e a chama. A função retorna um *Promise*, que em JavaScript, é um objeto usado para processamento assíncrono. Um *Promise* (promessa) representa um valor que pode estar disponível agora ou no futuro. Isso permite a associação de métodos de tratamento para eventos da ação assíncrona num caso eventual de sucesso ou de falha. Isto permite que métodos assíncronos retornem valores como métodos síncronos: ao invés do valor final, o método assíncrono retorna uma promessa ao valor em algum momento no futuro.

A Figura 17 apresenta a função *action* de um microserviço, que neste caso é chamado de "*math*". Para chamada interna dessa função, é preciso iniciar o microserviço via comando *broker.start*() e depois chamar ação via comando *broker.call*.

Figura 17 – Exemplo de chamada de ação "*math*" via *broker.call* de um microserviço do Moleculer.



```
const { ServiceBroker } = require("moleculer");

const broker = new ServiceBroker();

broker.createService({
    name: "math",
    actions: {
        add(ctx) {
            return Number(ctx.params.a) + Number(ctx.params.b);
        }
    }
});

broker.start()
    // Call service
    .then(() => broker.call("math.add", { a: 5, b: 3 }))
    .then(res => console.log("5 + 3 =", res))
    .catch(err => console.error(`Error occured! ${err.message}`));
```

Fonte: Moleculer (2018).

Métodos (*methods*): Os métodos são funções privadas do microserviço que somente são executadas internamente. Para criação de métodos no microserviço, os comandos devem ser colocados entre chaves dentro da função *methods*, como mostrado na Figura 18, e serem chamados a partir das funções de ação.

Figura 18 – Exemplo da propriedade *Methods* de um microserviço do Moleculer.

```
module.exports = {
    name: "mailer",
    actions: {
        send(ctx) {
            // Call the `sendMail` method
            return this.sendMail(ctx.params.recipients, ctx.params.subject, ctx.params.body)
        }
    },

    methods: {
        // Send an email to recipients
        sendMail(recipients, subject, body) {
            return new Promise((resolve, reject) => {
                ...
            });
        }
    }
};
```

Fonte: Moleculer (2018).

As funções privadas, não podem ser chamadas via *broker.call*. Para executar chamadas internas dos manipuladores de ações e de eventos, é necessário que se faça através da função *methods*.

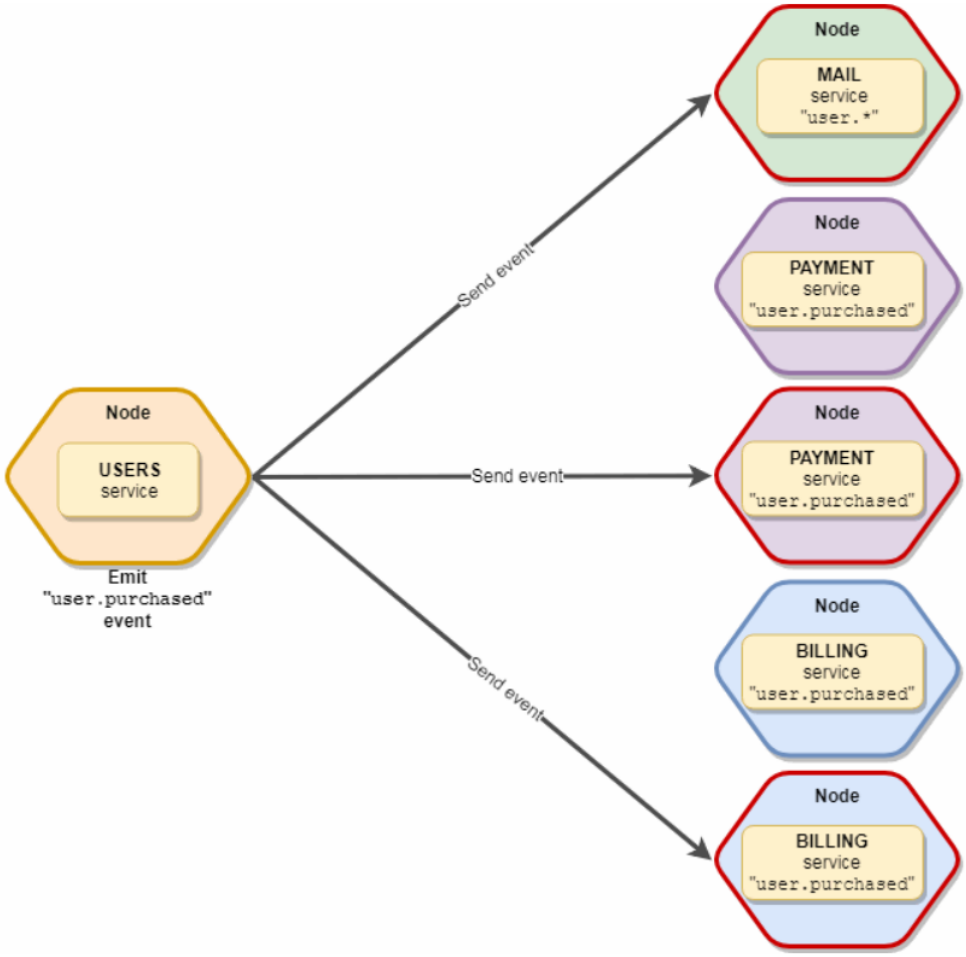
Eventos (*events*): O *ServiceBroker* possui um barramento integrado para suportar uma arquitetura orientada a eventos. Quando ocorrer um evento, uma mensagem pode ser enviada para os nós locais e remotos. A propriedade *events* (eventos) consiste de produtores de eventos que geram um fluxo de dados, e consumidores dos eventos que os escutam. Os eventos são entregues após a execução da propriedade, para que os consumidores possam responder imediatamente conforme a ocorrência desses eventos.

As arquiteturas de microserviços baseados em eventos usam o modelo de *publish/subscribe*. Quando um evento é publicado, ele é enviado para cada assinante. Depois que um evento é recebido, ele não pode ser reproduzido e não será exibido para assinantes novos. No Moleculer existem dois tipos de eventos como podem ser verificados:

- ✓ *Balanced events*;
- ✓ *Broadcast event*.

No caso dos eventos balanceados (*Balanced events*), os microserviços são organizados em grupos lógicos ouvintes dos eventos. Isso significa que apenas um microserviço ouvinte é acionado em cada grupo, como mostra a Figura 19.

Figura 19 – Diagrama de evento balanceado para microserviço no Moleculer.



Fonte: Moleculer (2018).

O nome do grupo (*group*) vem do nome do microserviço, porém pode ser definido na programação das propriedades do evento como é mostrado na Figura 20.

Figura 20 – Exemplo da propriedade do grupo do método evento de um microserviço do Moleculer.

```
module.exports = {
    name: "payment",
    events: {
        "order.created": {
            // Register handler to the "other" group instead of "payment" group.
            group: "other",
            handler(payload) {
                // ...
            }
        }
    }
}
```

Fonte: Moleculer (2018).

Para enviar um evento balanceado deve ser usada a função *broker.emit*. Nessa função o primeiro parâmetro é o nome do evento e o segundo parâmetro é o *payload* de acordo com a Figura 21.

Figura 21 – Exemplo da função *broker.emit* de um microserviço do Moleculer.

```
// The `user` will be serialized to transportation.
broker.emit("user.created", user);
```

Fonte: Moleculer (2018).

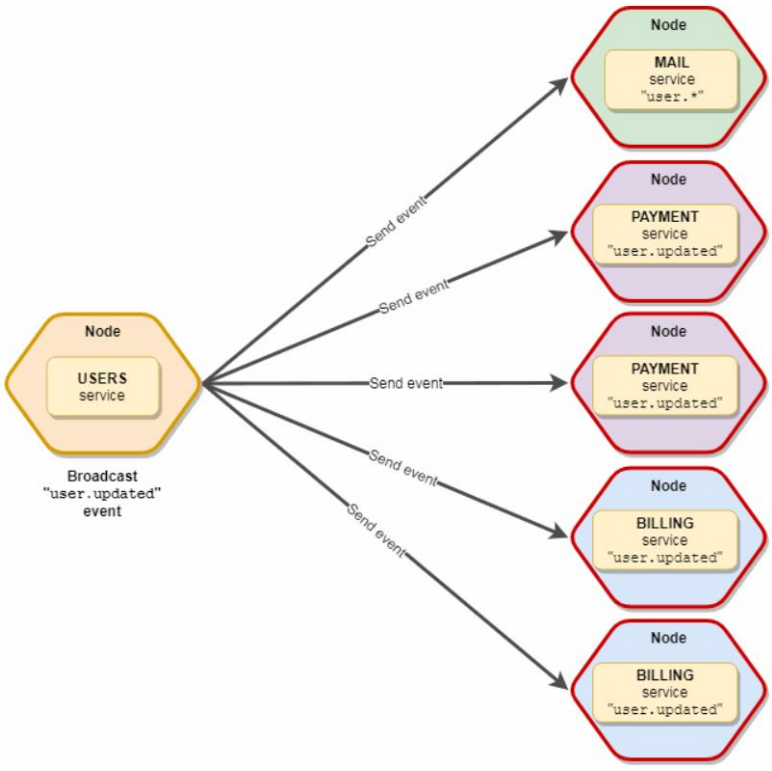
No *Broadcast event*, a transmissão dos eventos é enviada para todos os serviços locais e remotos disponíveis. A Figura 22 apresenta o método *broker.broadcast* para transmissão de eventos para todos os microserviços. A transmissão dos eventos nesse caso não é balanceada, sendo assim, todas as instâncias de serviço recebem o evento ocorrido como apresenta a Figura 23.

Figura 22 – Exemplo do método *broker.broadcast* de um microserviço do Moleculer.

```
broker.broadcast("config.changed", config);
```

Fonte: Moleculer (2018).

Figura 23 – Diagrama de um evento *Broadcast* de um microserviço no Moleculer.



Fonte: Moleculer (2018).

No Moleculer, todos os módulos principais possuem uma instância personalizada de um registrador de eventos relevantes à aplicação (*logs*). Também possui um registrador de console embutido que é o *logger* padrão (MOLECULER, 2018). A Figura 24 ilustra a configuração para a exibição dos *logs* da aplicação no console do computador e a exibição dos *logs* no console respectivamente.

Figura 24 – Exemplo de configuração de um *ServiceBroker* com registrador de *logs*.

```
[2018-06-26T11:38:06.728Z] INFO  node-100/POSTS: Log message via Service logger
[2018-06-26T11:38:06.728Z] INFO  node-100/BROKER: Log message via Broker logger
[2018-06-26T11:38:06.730Z] INFO  node-100/BROKER: ServiceBroker is stopped. Good bye.
```

```
const { ServiceBroker } = require("moleculer");
const broker = new ServiceBroker({
    nodeID: "node-100",
    // logger: true,
    logLevel: "info"
});

broker.createService({
    name: "posts",
    actions: {
        get(ctx) {
            this.logger.info("Log message via Service logger");
        }
    }
});

broker.start()
    .then(() => broker.call("posts.get"))
    .then(() => broker.logger.info("Log message via Broker logger"));
```

Fonte: Moleculer (2018).

3.3 *Gateway* (API)

O serviço *Gateway* (API) é responsável pela interface padronizada de conexão dos serviços do Moleculer com aplicações externas. Para acessar o microserviço *Gateway*, deve-se usar o IP do dispositivo na porta 3000. Alguns exemplos de URLs da API *Gateway* para chamada de ações ou informações disponíveis nos microserviços são descritos a seguir:

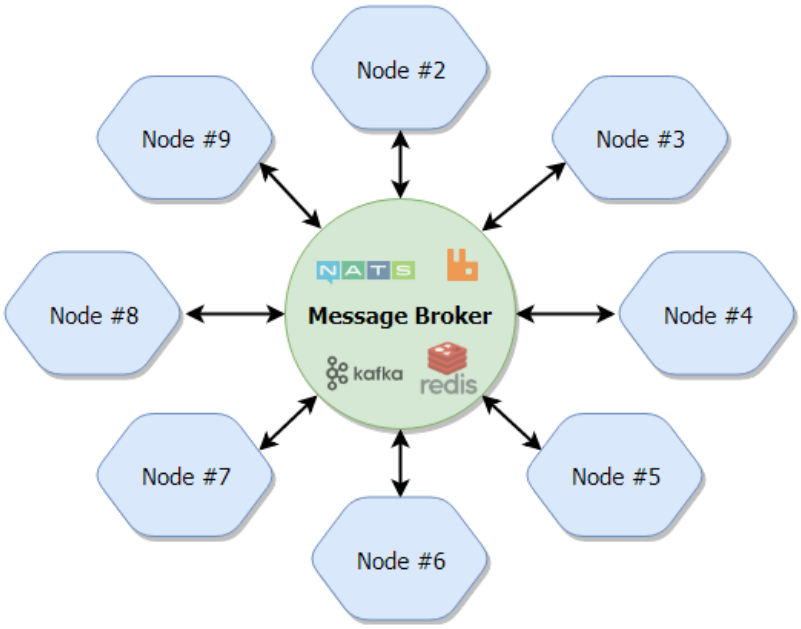
- ✓ Chamada da ação *hello* do microserviço teste: http://IP:3000/test/hello
- ✓ Chamada da ação *add* com parâmetros do microserviço *math*: http://IP:3000/math/add?a=25&b=13
- ✓ Chamada de informações do nó (microserviço) *health*: http://IP:3000/~node/health.

- ✓ Chamada de listagem das ações de todos nós (microserviços): http://IP:3000/~node/actions

3.4 *Transporter*

O serviço e mensagens (*Transporter)* é responsável pela de troca de mensagens entre os microserviços da arquitetura MOA, conforme é apresentado na Figura 25. O *framework* Moleculer disponibiliza uma série de mecanismos diferentes para comunicação entre os microserviços como: TCP/IP, NATS, MQTT, AMQP e etc.

Figura 25 – Microserviço *Transporter* da Arquitetura MOA.



Fonte: Moleculer (2018).

A configuração do tipo de *Transporter* é feita no método *ServiceBroker* do microserviço na opção *Transporter*. Após ser configurado o tipo de *broker* de mensagem (*Transporter*), a comunicação entre os microserviços ocorre de forma transparente, sendo assim, não importa qual o mecanismo de comunicação utilizado, não é necessário nenhum tipo de especificação ou configuração adicional para a comunicação entre os microserviços.

As chamadas internas das ações são executadas pela função *broker.call*. As chamadas externas das ações ocorrem através do microserviço *Gateway*. O padrão de troca de mensagens utilizado é o método *publish/subscribe*.

3.5 REPL console

O *moleculer-repl* é um console de desenvolvedor interativo para o Moleculer. Ele deve ser instalado pelo comando "*npm i moleculer-repl*" e usado no código, como mostra a Figura 26.

Figura 26 – Configuração do REPL no Moleculer

```
const broker = new ServiceBroker();

broker.start().then(() => {
    // Switch to REPL mode
    broker.repl();
});
```

Fonte: Moleculer (2018)

Através desse console é possível, por exemplo, obter ajuda (*help*) para um determinado comando, listar as ações dos microserviços, chamar uma ação, emitir um evento, listas métricas, listar os nós, listar os serviços, dentre outras várias opções, proporcionando facilidade para o gerenciamento dos microserviços. A Figura 27 mostra a lista de comandos do *REPL console*.

Figura 27 – Opções do REPL no Moleculer

```
Commands:
  help [command...]                                    Provides help for a given command.
  q                                                    Exit application
  actions [options]                                    List of actions
  bench [options] <action> [jsonParams]                Benchmark a service
  broadcast <eventName>                                Broadcast an event
  broadcastLocal <eventName>                           Broadcast an event locally
  call [options] <actionName> [jsonParams]             Call an action
  dcall [options] <nodeID> <actionName> [jsonParams]   Direct call an action
  clear [pattern]                                      Clear cache entries
  cls                                                  Clear console
  destroy <serviceName> [version]                      Destroy a locally running service
  emit <eventName>                                     Emit an event
  env                                                  List of environment variables
  events [options]                                     List of event listeners
  info                                                 Information about broker
  load <servicePath>                                   Load a service from file
  loadFolder <serviceFolder> [fileMask]                Load all services from folder
  metrics [options]                                    List metrics
  nodes [options]                                      List of nodes
  services [options]                                   List of services
```

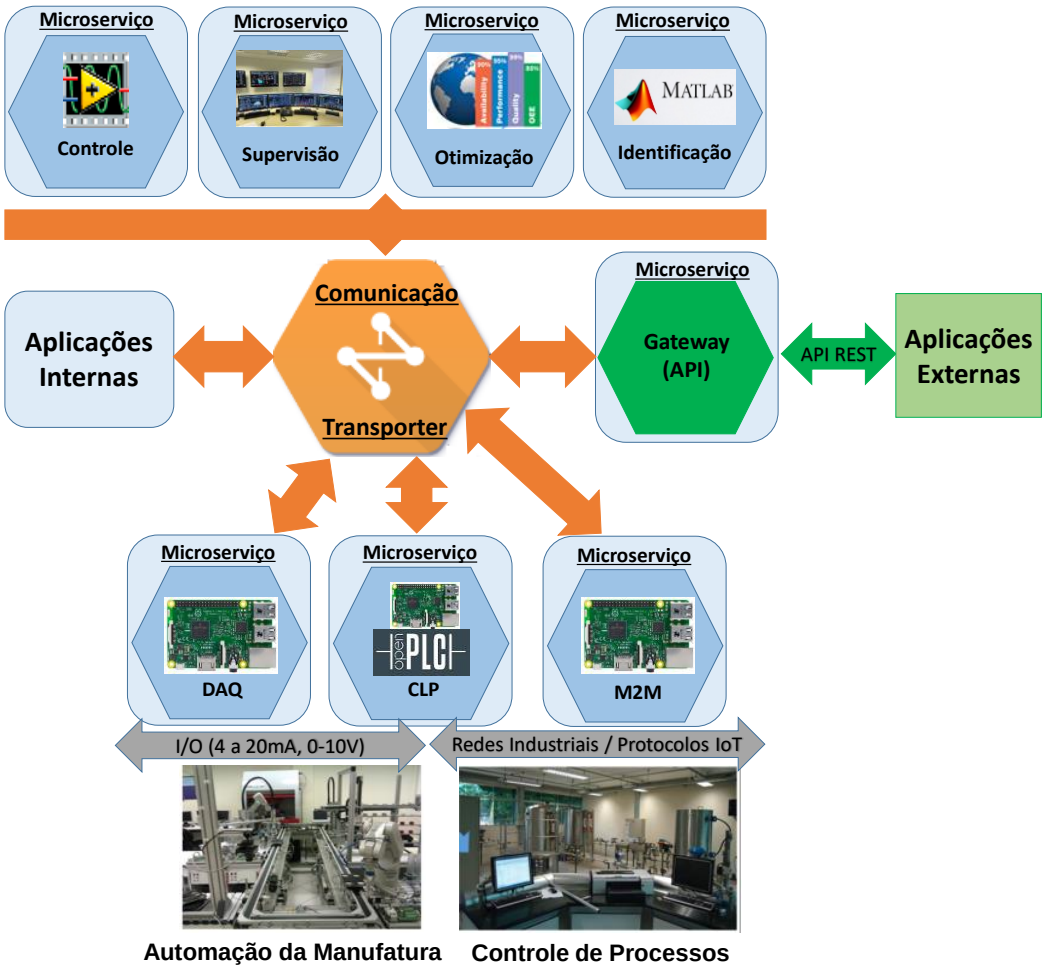
Fonte: Moleculer (2018)

4 PROPOSTA DO TRABALHO

Esse trabalho de Mestrado faz parte do desenvolvimento de um projeto de pesquisa financiado pela FAPESP (FAPESP 2018/19984-4). Nesse projeto de pesquisa é proposto o desenvolvimento de uma arquitetura orientada a microserviços (MOA) com base no *framework* Moleculer para aplicações de IIoT e I4.0, cuja estrutura pode ser vista na Figura 28.

O projeto apresenta o desenvolvimento de um conjunto de diferentes microserviços e aplicações para realização de atividades relacionadas à IIoT e I4.0. Este trabalho de Mestrado foca no desenvolvimento dos serviços Controlador Lógico Programável (CLP), serviço Aquisição de Dados (DAQ) e serviço Controle PIDPlus, além de aplicações de Controle e Supervisão.

Figura 28 – Diagrama da Arquitetura MOA com *framework* Moleculer para Aplicações de IIoT e I4.0.



Fonte: Projeto Fapesp 2018/19984-4 (https://bv.fapesp.br/pt/auxilios/103005).

Os microserviços podem ser executados em três tipos de plataforma, dependendo de cada aplicação e/ou necessidade: sistema embarcado (ex: Raspberry Pi 3B+) com sistema

operacional Linux, um computador (PC) com sistema operacional (Windows, Linux, MAC) e uma plataforma em nuvem comercial ou industrial (local). Os serviços possíveis de operar em nuvem funcionariam numa estrutura de PaaS (*Platform as a Service*).

Os serviços da arquitetura MOA da Figura 28 oferecem uma funcionalidade principal, podendo executar diferentes ações, métodos e eventos relacionados àquela funcionalidade. A comunicação dessa estrutura tem como base o protocolo Ethernet TCP (UDP)/IP, que possibilita grande interconectividade entre os componentes.

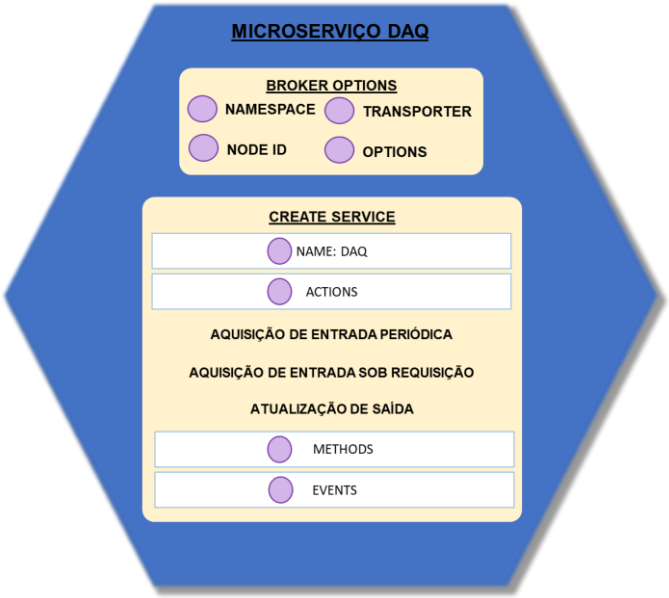
4.1 Microserviço DAQ

O microserviço DAQ é responsável por realizar a aquisição e atualização de dados das variáveis de interesse por meio de um *hardware* dedicado alocado no processo, que também pode ser implementado em dois tipos de plataforma: computador (PC) ou sistema embarcado. A atividade de aquisição e atualização de dados é perceptível aos outros microserviços ou aplicações que utilizam o microserviço DAQ, de forma que a especificação de *hardware* e *software* necessários para a aquisição/atualização dos dados das variáveis de interesse são irrelevantes aos outros microserviços que compõem a arquitetura.

Sendo assim, o microserviço DAQ poderia ser implementado em um computador (PC), e se comunicar via USB ou PCI com alguma placa ou módulo DAQ para realização das atividades de aquisição/atualização das variáveis, por exemplo. Outra possibilidade seria rodar em um sistema embarcado (Raspberry Pi) e usar um *hardware* externo ou conexão de I/O direta para realização das atividades de aquisição/atualização das variáveis de interesse. Portanto, resumidamente, para uma a aplicação que usa o microserviço DAQ, não importa qual *hardware* e/ou *software* estão sendo usados para aquisição/atualização das variáveis e sim somente a informação/dado das variáveis.

De acordo com a Figura 29, o microserviço DAQ é basicamente composto de ações e eventos. As ações estão relacionadas às funcionalidades principais: aquisição de uma entrada e atualização de uma saída (canais de I/O). A aquisição de dados de uma entrada pode ser realizada de duas formas: periódica, onde o envia o dado da variável via *transporter* de acordo um tempo cíclico predeterminado, e sob requisição, onde diante do recebimento de uma requisição, o microserviço envia via *transporter* o último dado coletado da variável de interesse. É importante destacar que as ações do microserviço podem ser atreladas a diferentes tipos de I/O, como entradas e saídas analógicas e digitais, sendo que a única diferença é o tamanho do dado (em *bits*) da variável de interesse, sendo 1 bit para variáveis digitais e 16 *bits* para variáveis analógicas.

Figura 29 – Microserviço DAQ.

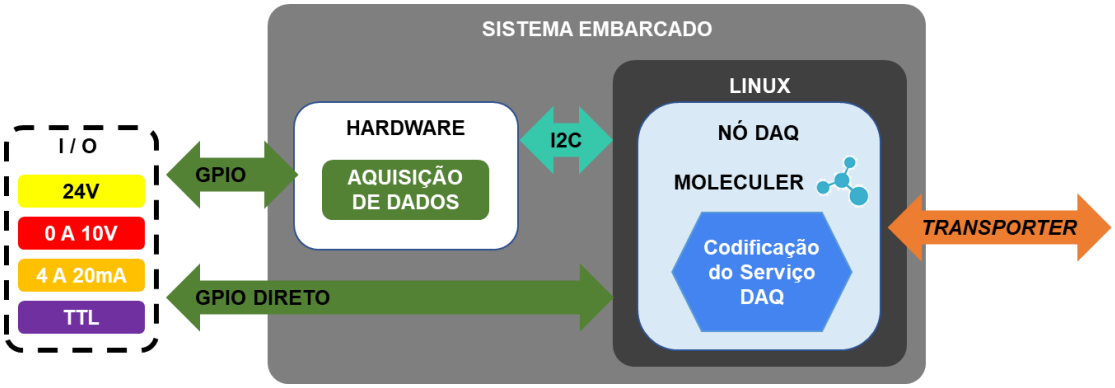


Fonte: Autor.

O sistema DAQ desenvolvido neste trabalho é baseado em um circuito de entradas e saídas digitais e analógicas, construído a partir de um dispositivo central que coleta e envia informações. A Figura 30 mostra um esquemático geral de como funciona o sistema, onde no *hardware* externo temos os circuitos condicionadores de sinais, que recebem sinais analógicos e digitais, que são convertidos em níveis de 0 a 5V.

Esses sinais são recebidos pelo *hardware* principal, que disponibiliza os dados de tal forma que outros microserviços possam solicitar as informações/dados do sistema DAQ. Os dados coletados são condicionados no nó DAQ, que contém a codificação do servidor com as configurações necessárias. O *Transporter* é o responsável por viabilizar a comunicação entre os microserviços, e pode estar operando em qualquer outro computador, desde que estejam na mesma rede, como é o caso apresentado nesse trabalho.

Figura 30 – Estrutura de Implementação do Nó e Serviço DAQ.



Fonte: Autor.

4.2 Microserviço Controle PIDPlus

O serviço Controle PIDPlus funciona como um controlador de processos, onde a partir de um algoritmo de controle, efetua o cálculo de um sinal a ser aplicado no processo que se deseja controlar. Cada instância (réplica) desse serviço é responsável pela implementação de uma malha de controle fechado. Este serviço pode ser implementado em três tipos de plataforma: computador (PC), sistema embarcado ou nuvem. Para que seja possível obter as variáveis do processo a ser controlado, este serviço necessita operar em conjunto com os serviços DAQ e/ou M2M.

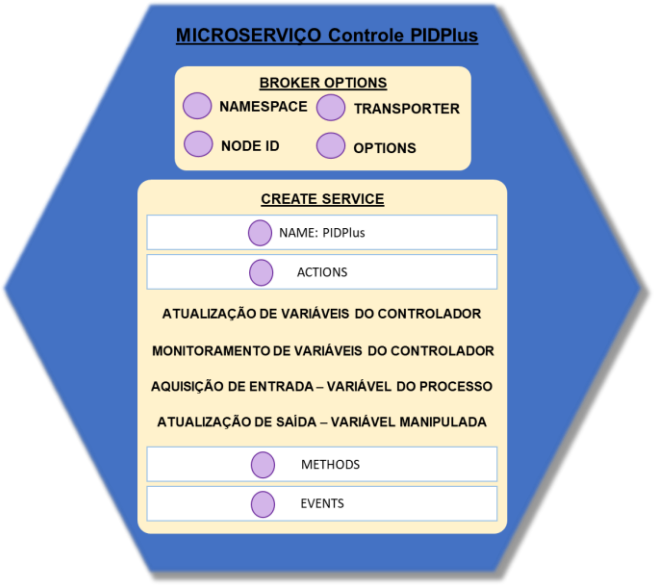
Um pressuposto em controle de processo sempre foi que sua execução se dará em uma base periódica e que um novo valor de medição está disponível para cada execução. No entanto, o PIDPlus tem a sua base de execução aperiódica baseada em requisições, e a maior parte do atraso dessas requisições no microserviço PIDPlus é absorvida dentro da execução da malha tendo efeito reduzido no loop de controle.

No entanto, o atraso gerado nas requisições pode ocasionalmente mudar de forma aleatória, devido à natureza dinâmica da rede. Quando a medição não é atualizada em uma base periódica, um PID tradicional não é indicado, pois as ações de controle não serão calculadas de forma correta. Como o controle só é executado quando uma nova medida é informada, isso poderia resultar em um atraso na resposta de um controlador.

Para absorver estes atrasos gerados de forma aleatória pelas requisições na rede, o serviço Controle utiliza como base o algoritmo de controle PIDPlus (SONG et al., 2006; BLEVINS e al., 2014) para cálculo das ações de controle. Este algoritmo foi criado para aplicações de controle em malha fechada usando redes de comunicação, mas neste estudo proposto foi modificado para trabalhar com o conceito de requisições e operar como um serviço (BIGHETI, FERNANDES & GODOY, 2019). Embora esse serviço use especificamente o algoritmo de controle PIDPlus, um ponto importante a se destacar é que seria possível criar outros microserviços de controle que implementassem outros algoritmos e técnicas de controle diferentes, fornecendo flexibilidade e modularidade para o desenvolvimento e implementação.

A Figura 31 demonstra que o microserviço Controle PIDPlus possui basicamente ações, e essas ações estão relacionadas às funcionalidades principais do serviço: Atualização de variáveis do controlador (*Setpoint* e Parâmetros de configuração do controlador como ganho proporcional - K, Tempos integrativo - Ti e Derivativo – Td, Constante DeltaT), Monitoramento de variáveis do controlador (*Setpoint*, Variável do processo e Variável Manipulada), Aquisição de entrada - variável do processo (PV – *Process Variable*) e Atualização de saída – variável manipulada (MV – *Manipulated Variable*).

Figura 31 – Microserviço Controle PIDPlus.

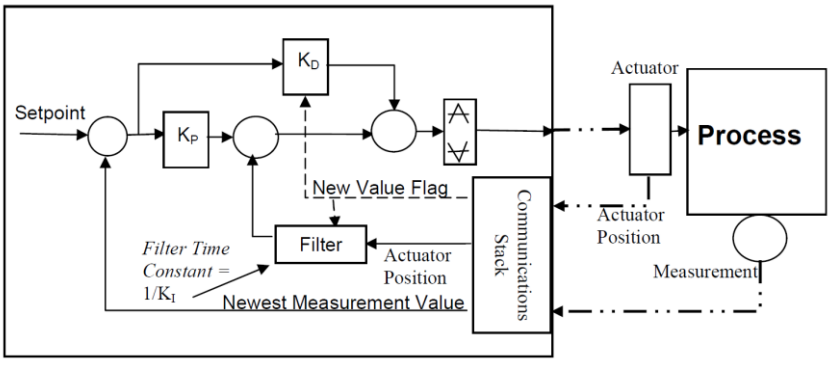


Fonte: Autor.

4.2.1 Algoritmo de Controle PIDPlus

O PIDPlus é um controlador PID modificado para operar de forma adequada com grandes períodos de amostragem do sensor, atualizações não periódicas da variável e perda de mensagens transmitidas (BLEVINS et al., 2014). A implementação PIDPlus (SONG et al., 2006) é ilustrada na Figura 32, que mostra que o *reset* das partes integrais/derivativas do PID são determinadas com base no tempo entre recepções de mensagens (*New Value Flag*) pela pilha de comunicações.

Figura 32 – Estrutura do controlador PIDPlus.



Fonte: SONG et al. (2006).

O PIDPlus mantém o sinal de controle no último nível calculado até que uma nova medida seja recebida. Uma observação importante é que sua sintonia é independente do período de amostragem, dependendo apenas das características físicas da planta. A realimentação (*Newest*

Measurement Value) e o filtro de 1ª ordem (*Filter*) são modificados para criar a contribuição de reposição com o seguinte comportamento:

- ✓ Manter a última saída do filtro calculado (F_{N-1}), até uma nova medição ser informada (*New Value Flag*);
- ✓ Quando uma nova medição é recebida (*New Value Flag*), utilize a nova saída do filtro como contribuição da realimentação (F_N).

A principal diferença do PID e PIDPlus está na parte integrativa que é substituída por um filtro de 1ªordem. Para contabilizar a resposta do processo, a saída do filtro é calculada conforme equação (1) quando uma nova medição é recebida:

$$F_N = F_{N-1} + (O_{N-1} - F_{N-1}) * \left(1 - e^{\frac{-\Delta T}{T_{reset}}}\right) \quad (1)$$

onde: F_N = nova saída do filtro, F_{N-1} = saída do filtro na última execução, O_{N-1} = saída do controlador na última execução, ΔT = tempo decorrido desde que o último valor medido foi recebido, T_{reset} = constante de tempo da planta somado ao tempo morto.

A parte derivativa do algoritmo modificado (limitação de taxa não aplicada) é determinada pela seguinte equação (2):

$$O_D = K_D * \frac{e_N - e_{N-1}}{\Delta T} \quad (2)$$

onde: O_D = termo derivativo do controlador, K_D = ganho derivativo, e_N= erro atual, e_{N-1} = último erro, ΔT = tempo decorrido desde que o último valor medido foi recebido.

Considere a contribuição da parte derivativa quando as entradas forem perdidas por vários períodos. Quando a comunicação é restabelecida, e_N - e_{N-1} na equação (2) seria o mesmo para os algoritmos originais e modificados. No entanto, para o algoritmo PID padrão, o divisor na parte derivada seria o período, enquanto que no novo algoritmo é o tempo decorrido entre duas medições recebidas com sucesso, sendo assim o algoritmo modificado produziria uma ação derivada menor que o algoritmo PID padrão (SONG et al., 2006).

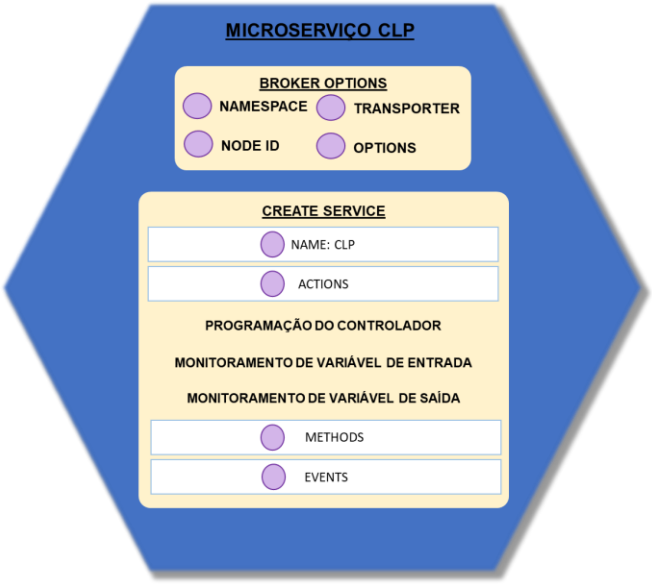
Na implementação do PIDPlus, o cálculo de reposição compensa automaticamente a alteração da medição e taxa de atualização da medição. Os cálculos do termo derivativo para um novo valor de medição não estão disponíveis a cada execução do PID. Assim, não há necessidade de modificar a sincronização para o controle sem fio, ou seja, o ajuste é baseado estritamente no ganho e dinâmica do processo.

4.3 Microserviço CLP

O microserviço CLP é responsável por integrar as funcionalidades de um controlador lógico programável (CLP) na arquitetura do Moleculer. A proposta é adicionar um SoftPLC como serviço na estrutura do Moleculer, que basicamente é um recurso que une as funcionalidades de um CLP com os benefícios de uma arquitetura aberta de *hardware* e *software*. O objetivo desse serviço é executar lógicas e sequenciamento de tarefas que são executadas em CLPs convencionais, bem como o monitoramento de dados de entradas e saídas digitais e analógicas do controlador, porém sendo implementado em um sistema embarcado dedicado alocado no processo.

De acordo com a Figura 33, o microserviço CLP possui basicamente ações, relacionadas às funcionalidades principais do serviço: Programação do controlador, Monitoramento de variável de entrada e Monitoramento de variável de saída. As ações de monitoramento de I/O do microserviço podem ser atreladas a diferentes tipos de I/O, como entradas e saídas analógicas e digitais, sendo que a única diferença é o tamanho do dado (em *bits*) da variável de interesse, sendo 1 *bit* para variáveis digitais e 16 *bits* para variáveis analógicas.

Figura 33 – Microserviço CLP.



Fonte: Autor.

Para o desenvolvimento desse microserviço foi necessário promover a integração do projeto OpenPLC (OPEN PLC, 2019) junto à arquitetura de microserviço do Moleculer usada.

O projeto OpenPLC segue as premissas da norma IEC 61131[1] e da PLCOpen XML[2], permitindo a programação de *hardware* livre de acordo com o padrão IEC 61131-3, o qual define cinco linguagens padrão para programação de CLPs: Diagrama de Blocos Funcionais (FBD - *Function Block Diagram*), Diagrama *Ladder* (LD - *Ladder Diagram*), Texto Estruturado (ST - *Structured Text*), Lista de Instruções (IL - *Instruction List*), Sequenciamento Gráfico de Funções (SFC - *Sequential Function Chart*). A programação do *hardware*/CLP em uma das linguagens do padrão IEC 61131-3 é feita por meio do *OpenPLC Editor*.

4.4 Aplicações Internas e Externas

O *framework* Moleculer fornece uma estrutura padronizada que viabiliza e simplifica a estruturação dos microserviços desenvolvidos, que podem conter Aplicações Internas e Externas. O Moleculer permite que essas aplicações sejam estruturadas de formas que os microserviços estejam alocados de duas formas: Distribuída (Figura 9) e Mista (Figura 10).

No caso das aplicações distribuídas com microserviços, todos os serviços são executados em nós individuais e se comunicam via *Transporter*. Nesse caso, a latência da rede não é desprezível. Mas os serviços podem ser dimensionados/replicados para serem resilientes e tolerantes a falhas (MOLECULER, 2018). No caso das aplicações mistas com microserviços, serviços coerentes podem ser agrupados e executados no mesmo nó. Esse tipo de recurso pode ser utilizado para reduzir a latência da rede entre dois ou mais microserviços que realizam chamadas entre si por várias vezes.

Apesar de a arquitetura do Moleculer permitir a criação das Aplicações Internas e Externas através da composição de serviços por Orquestração e por Coreografia, neste trabalho foi definido o uso da Orquestração para criação das aplicações.

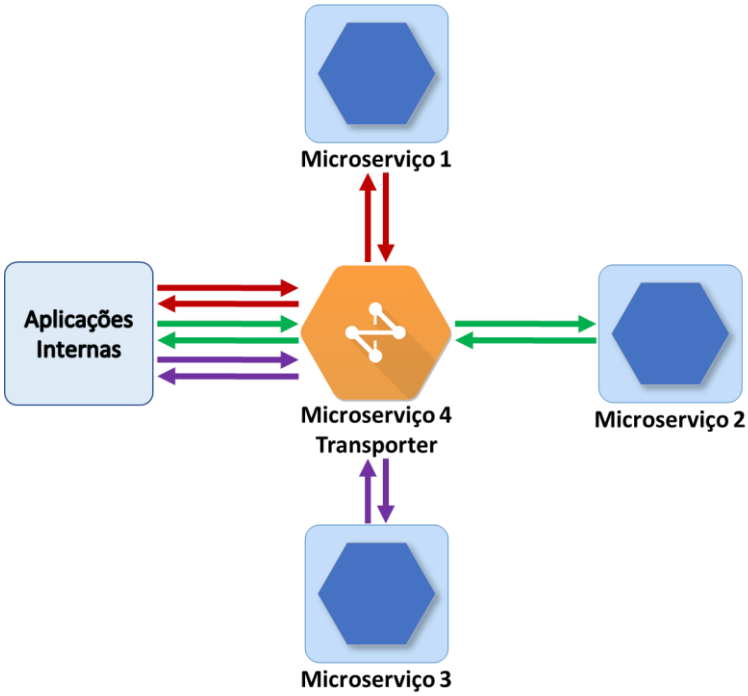
As Aplicações Internas correspondem às aplicações que são integralmente desenvolvidas nas plataformas suportadas pelo Moleculer (JavaScript - Node.js, Java, .NET e Go – MOLECULER, 2018). Nesse tipo de aplicação, a orquestração dos microserviços é feita diretamente usando somente comunicação via microserviço *Transporter*, de forma que a sequência dos serviços é gerenciada usando o formato de comunicação requisição/resposta.

A Figura 34 apresenta um exemplo de orquestração de microserviços de uma Aplicação Interna usando o Moleculer, composta por 4 microserviços,

[1] IEC 61131 é um padrão internacional para controladores programáveis industriais, contemplando o projeto completo desde *hardware*, instalação, testes, documentação, programação e comunicação.

[2] PLCOpen XML é um padrão aberto e não proprietário para especificação em formato XML de programas de controladores programáveis padronizados pela IEC 61131-3, para integração com outras ferramentas.

Figura 34 – Aplicação Interna do Moleculer por Orquestração de Serviços.



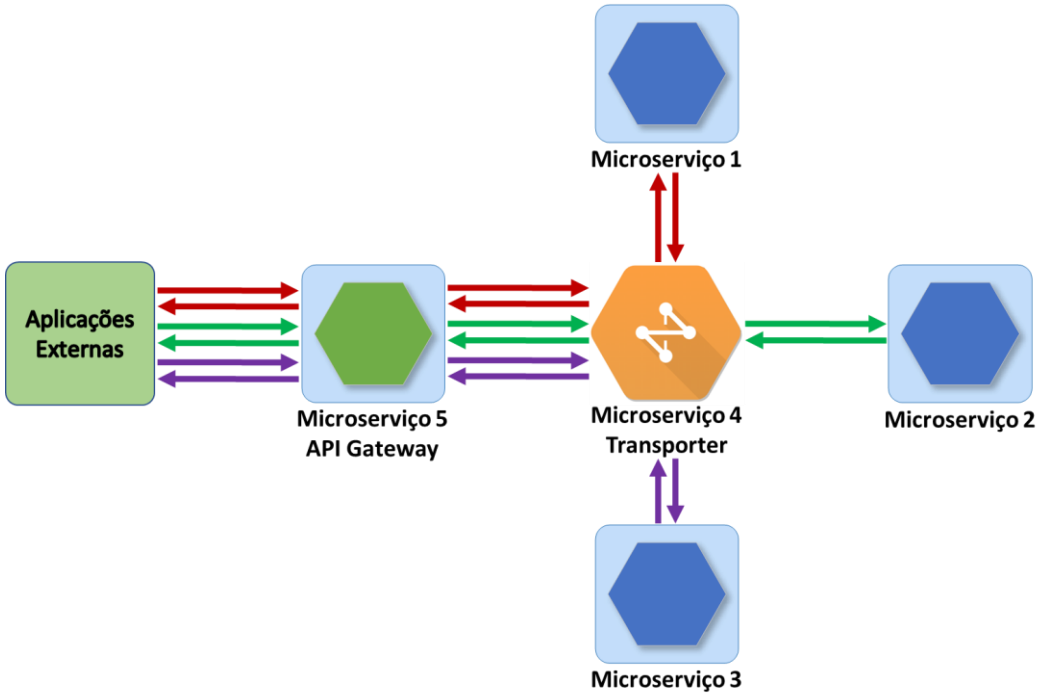
Fonte: Autor.

De acordo com a Figura 34, a Orquestração define a execução dos microserviços na sequência {4,1,4,4,2,4,4,3,4}. Nessa configuração, os microserviços 1, 2 e 3 estão alocados em Nós diferentes do Moleculer e, devido a isso, eles sempre utilizam o microserviço 4 (*Transporter)* para estabelecer a comunicação e integração entre si. É possível verificar que na orquestração, a Aplicação Interna é o gerenciador (maestro) da execução dos outros microserviços. Cada serviço é requisitado, e após a resposta desse serviço, a Aplicação requisita a execução de um próximo serviço e assim sucessivamente.

As Aplicações Externas correspondem às aplicações que são desenvolvidas em qualquer plataforma de desenvolvimento diferente das suportadas pelo Moleculer. Neste tipo de aplicação, a orquestração dos serviços ou sequência de operações definida sempre necessita ser realizada por meio de uma aplicação externa usando comunicação REST via microserviço *Gateway*.

De um modo geral, a estrutura utilizada para as Aplicações Externas é a mesma utilizada para as Aplicações Internas, o que as diferencia é o uso obrigatório do microserviço *Gateway* que viabiliza a interação entre os microserviços internos com a Aplicação Externa. A Figura 35 apresenta um exemplo de orquestração de microserviços de uma Aplicação Externa usando o Moleculer, composta por 5 microserviços.

Figura 35 – Aplicação Externa no Moleculer por Orquestração de Serviços.



Fonte: Autor.

Nessa configuração da Figura 35, a Aplicação Externa se comunica com os microserviços do Moleculer por meio do microserviço 5 *Gateway*, que por sua vez se comunica com os demais microserviços através do *Transporter*, conforme estrutura apresentada. A orquestração define a execução dos microserviços na sequência {5,4,1,4,5,4,2,4,5,4,3,4,5}. Da mesma forma que no exemplo anterior, os microserviços 1, 2, 3 e 5 estão alocados em Nós diferentes do Moleculer e, devido a isso, eles sempre utilizam o microserviço 4 (*Transporter*) para estabelecer a comunicação e integração entre si. É possível verificar que na orquestração, a Aplicação Externa é o gerenciador (maestro) da execução dos outros microserviços. Cada serviço é requisitado, nesse caso sempre passando pelo microserviço *Gateway*, e após a resposta desse serviço, a Aplicação requisita a execução de um próximo serviço, e assim sucessivamente.

É importante compreender o funcionamento das arquiteturas que podem ser implementadas no Moleculer, para que se possa construir cada aplicação da maneira mais adequada.

5 DESENVOLVIMENTO DOS SERVIÇOS E RESULTADOS

Este capítulo descreve sobre o desenvolvimento dos microserviços propostos nas plataformas utilizadas e apresenta as aplicações criadas a partir da coreografia dos serviços. No primeiro experimento foram implementados e testados microserviços com as funções de aquisição de dados de entrada e saídas digitais e analógicas. O segundo experimento é com a implementação do controle de velocidade de um motor CC de uma planta didática utilizando os microserviços DAQ, *Transporter*, *Gateway* e Controle PIDPlus. O terceiro experimento engloba o controle de uma planta industrial virtual e o controle em malha fechada da planta didática, utilizando os microserviços CLP, *Transporter* e a aplicação interna de Monitoramento.

A implementação dos códigos em JavaScript dos microserviços foi desenvolvida por meio do *software* Visual Studio Code. A extensão SSHExtension instalada no Visual Studio, permite que o console da Raspberry Pi seja acessado remotamente, ou seja, toda a construção do código é feita pelo PC, não sendo necessário o uso de periféricos como teclado, mouse e monitor conectados à Raspberry. Outro detalhe importante é que foi criada uma pasta compartilhada no sistema operacional Raspbian, instalado no dispositivo embarcado (Raspberry Pi). Essa pasta facilita o gerenciamento dos arquivos, uma vez que, o acesso pode ser feito através do PC (Windows). Dessa forma, todo o desenvolvimento das aplicações fica centralizado em um único computador.

A inicialização automática dos serviços em conjunto com o sistema operacional pode ser feita utilizando um gerenciador de processos de produção para aplicativos Node.js com um balanceador de carga interno denominado PM2. Ele permite manter os aplicativos ativos para sempre, recarregá-los sem tempo de inatividade e facilitar tarefas comuns de administração do sistema (NPM, 2020).

Este trabalho não abordou o desenvolvimento de soluções de segurança ao acesso dos microserviços, pois o objetivo central era verificar o desempenho e funcionamento das aplicações desenvolvidas. Porém, algumas possibilidades relacionadas a esse tema são possíveis de se implementar com a arquitetura MOA baseada no Moleculer, como por exemplo, criptografia HTTPs, autenticação de usuário e controle de acesso aos microserviços.

5.1 Desenvolvimento do Microserviço DAQ

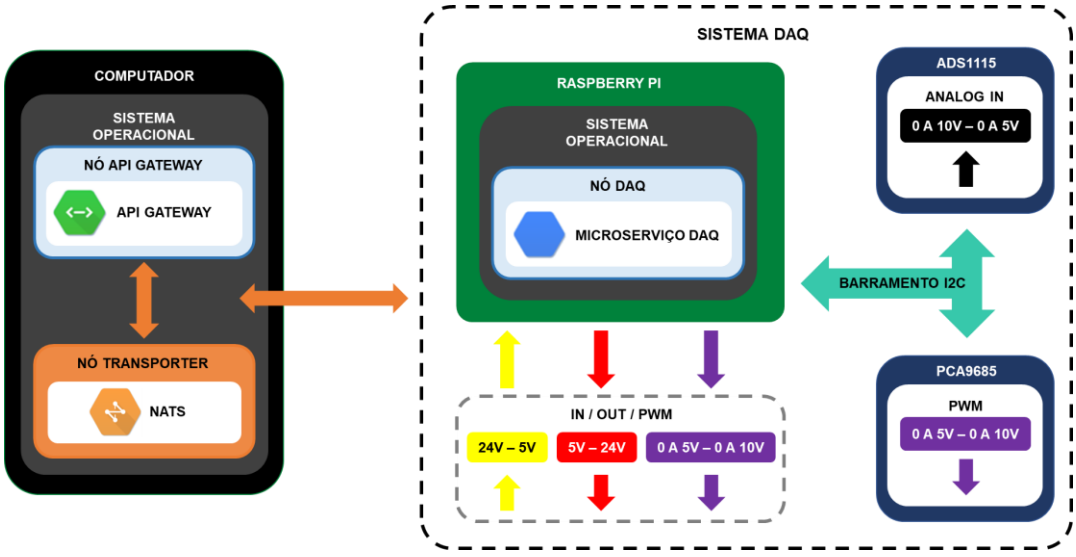
Para o desenvolvimento do microserviço DAQ foi implementada uma arquitetura composta de dois microserviços, onde um desses microserviços é um API *Gateway* e outro é um DAQ. O funcionamento desse sistema consiste basicamente de requisições feitas ao

servidor do microserviço DAQ pelo API *Gateway*, que realiza essas requisições via navegador *web*, que por sua vez são respondidas de acordo com cada solicitação. Por exemplo, se no sistema DAQ existe um sensor de temperatura conectado, e se deseja saber qual a temperatura que ele está submetido em um determinado instante, através do API *Gateway* é feita uma requisição via navegador *web* ao servidor DAQ, que retornará o dado solicitado no momento da requisição.

Tanto o servidor API *Gateway* quanto o servidor DAQ foram desenvolvidos sobre a plataforma Node.js, utilizando o *framework* Moleculer. O Node.js é instalado no sistema operacional de cada máquina no caso apresentado, que como ilustra a Figura 36, são duas maquinas distintas. No caso da Raspberry Pi, que compõe o sistema DAQ, o sistema operacional utilizado é o Raspbian, que é uma plataforma Linux especifica para o uso desse dispositivo embarcado. Já no computador, o sistema operacional utilizado foi o Windows, o qual também foi instalado o Node.js.

A Figura 36 também mostra a configuração do sistema de um modo geral, que consiste de um computador pessoal com o sistema operacional Windows, como já foi dito anteriormente, e nesse sistema operacional está o servidor com o microserviço API *Gateway* e também o *Transporter* NATS. A outra parte é constituída de uma placa Raspberry Pi 3 conectada via barramento I2C com um *shield* de 4 entradas ADC (ADS1115) e com um outro *shield* de 16 saídas PWM (PCA9685). Também é possível observar que existem condicionadores de sinais para as entradas e saídas digitais e analógicas, isso é porque a Raspberry Pi e os *shields* não trabalham com sinais de padrão industrial diretamente, sendo necessário a utilização de circuitos adicionais que condicionem esses sinais elétricos.

Figura 36 – Sistema DAQ.



Fonte: Autor.

Cada parte do sistema é considerado como um nó dentro do contexto de microserviço do Moleculer, sendo que API *Gateway*, *Transporter* e Microserviço DAQ são nós que interagem entre si dentro da arquitetura de microserviços. Para que isso se torne possível, cada microserviço deve ser configurado adequadamente para que corresponda as solicitações, que nesse caso, são feitas através de um navegador *web*.

Os microserviços no Moleculer executam ações por meio de “rotas” que são inseridas na barra de endereços do navegador. Basicamente o que acontece é que cada ação é executada por um endereço de uma página *web*, como por exemplo, retornando ao exemplo do sensor de temperatura citado anteriormente, para solicitar ao servidor DAQ o valor da temperatura que o sensor está submetido em determinado instante, é necessário acessar a página que contém a rota da ação da leitura do sinal desse sensor, algo semelhante a inserir um endereço do tipo:

<http://localhost:3000/sensores/temperaturas>

Para melhor entender o endereçamento das ações, podemos analisar o trecho de um código de um microserviço do Moleculer.

Figura 37 – Trecho de código JavaScript de um microserviço do Moleculer.

```
broker.createService({ // Cria o serviço
    name: "onoff", // Nome do serviço
    actions: { // Ações do serviço
        led(ctx){
```

Fonte: Autor.

A composição do endereço da ação do código JavaScript apresentado segue a seguinte sequência:

- ✓ *localhost* seguido da porta 3000, supondo que esteja usando o servidor API *Gateway* no mesmo computador de onde será feito o acesso através do navegador. No caso de acesso por um dispositivo externo, se utiliza o IP do computador onde está o serviço seguido da porta 3000.
- ✓ O nome do serviço, que se encontra dentro das chaves do *broker.createService*, que no código é nomeado como "*onoff* ".
- ✓ A identificação ou nomenclatura da ação, está dentro das chaves do *actions*, descrita como *led(ctx).*

Portanto, nesse caso o endereço para acessar a rota dessa ação seria o seguinte: <http://localhost:3000/onoff/led>.

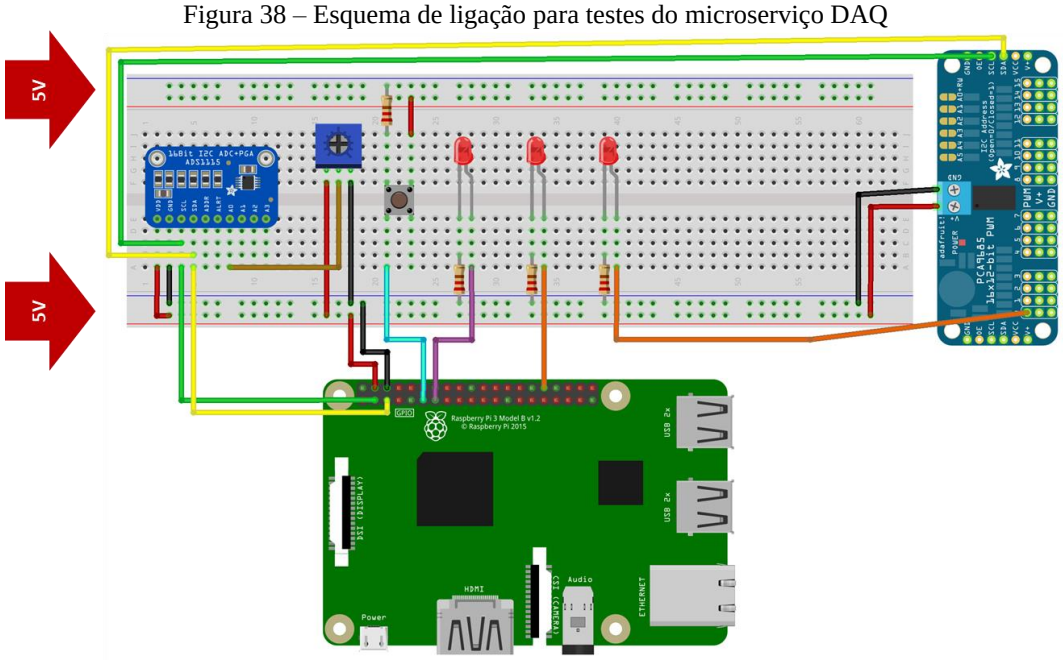
O microserviço DAQ é basicamente composto de 4 tipos de ações: uma aquisição de entrada periódica que atualiza o valor ciclicamente; uma aquisição de entrada sob requisição que acessa o valor no instante em que a requisição é realizada; uma aquisição de entrada por mudança de estado ou valor que dispara um valor sempre que houver uma alteração na leitura; atualização das saídas digitais e PWM.

Os testes realizados para verificar o funcionamento do microserviço DAQ foram:

- ✓ Acionar uma entrada digital da Raspberry Pi 3;
- ✓ Acionar uma saída digital da Raspberry Pi 3;
- ✓ Acionar uma saída PWM da Raspberry Pi 3;
- ✓ Realizar a leitura de um sinal analógico por um conversor ADC com comunicação I2C;
- ✓ Acionar as saídas PWM via comunicação I2C.

A Figura 38 apresenta o esquema de ligação utilizado nos testes. Todos esses acionamentos foram divididos em ações, sendo uma ação para cada tipo de leitura de sinal ou acionamento de saída, de forma que, para acessar cada ação, uma requisição deve ser feita via navegador para o servidor, que retorna com o resultado da solicitação.

Figura 38 – Esquema de ligação para testes do microserviço DAQ



Fonte: Autor

Para observar o comportamento do sistema DAQ durante os testes, foram conectados LEDs em uma saída digital e em uma saída PWM da Raspberry Pi e também em uma saída PWM do modulo PCA9685. Um botão de pulso foi conectado em uma entrada digital da Raspberry e um potenciômetro foi conectado em uma entrada do conversor ADC. A atualização dos sinais das saídas e das entradas foram configuradas separadamente, portanto, esse sistema permite acessar qualquer uma dessas ações de forma independente de acordo com a necessidade do usuário.

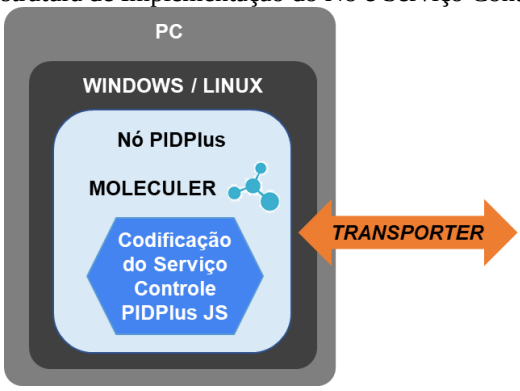
5.2 Desenvolvimento do Microserviço Controle PIDPlus

Para compor a arquitetura MOA deste trabalho foi desenvolvido um controlador de processo no formato de um microserviço. Isso requer uma nova concepção ou a adaptação de um algoritmo de controle tradicional para o formato de serviço. Como a arquitetura MOA é intrinsicamente uma arquitetura conectada em rede, é adequado propor o uso de um algoritmo de controle em rede neste trabalho.

O controlador PIDPlus citado na seção 4.2 é uma versão modificada do controlador PID tradicional para aplicações de controle em malha fechada. Este controlador é um microserviço responsável pelo controle de processos, onde o acesso é feito por requisições na rede. O algoritmo de controle base PIDPlus (seção 4.2.1) foi alterado para operar com base em requisições do *framework* Moleculer e implementado em linguagem JavaScript. Dessa forma, o controlador desenvolvido (em JavaScript) pode ser utilizado como um serviço do Moleculer e requisitado por qualquer outro serviço ou aplicação (Interna ou Externa).

O microserviço Controle PIDPlus desenvolvido neste trabalho possui a estrutura mostrada na Figura 39. Este microserviço opera em um computador com sistema operacional Windows ou Linux.

Figura 39 – Estrutura de Implementação do Nó e Serviço Controle PIDPlus.



Fonte: Autor.

Inicialmente, o algoritmo PIDPlus, devido à algumas particularidades de implementação (como a necessidade do cálculo do tempo entre recebimento de mensagens com o dado da variável do processo) foi utilizado em um servidor desenvolvido no *software* LabVIEW (BIGHETI et al., 2018). Essa estrutura é interessante, pois flexibiliza o uso de um código desenvolvido fora do Moleculer dentro da estrutura de um serviço. No entanto, há a necessidade e uma comunicação adicional padrão REST entre o código do serviço e o código rodando no LabVIEW como um servidor. Com o objetivo de melhorar o desempenho de execução do serviço Controle PIDPlus, este foi desenvolvido para compor a estrutura de microserviços do Moleculer.

O PIDPlus implementado em JavaScript contém uma ação dentro do microserviço que recebe os parâmetros de ganho proporcional (kp), tempo integrativo (ti), tempo derivativo (td), *setpoint*, variável do processo (PV) e a variação de tempo do processo (ΔT). Essa ação é acessada por meio de uma rota, ou seja, um endereço por onde são enviados dados das variáveis citadas anteriormente utilizando a arquitetura REST. A atualização desses dados é essencial para o funcionamento do controlador que utiliza como base o algoritmo citado na seção 4.2.1.

Para armazenar os dados do estado anterior das variáveis de erro (e_{N-1}), de saída (O_{N-1}) e do termo integrativo (F_{N-1}) do controlador foi criado um *shift register*. O *shift register* é basicamente um dispositivo sequencial que carrega os dados atuais em suas entradas e, em seguida, os desloca para sua saída uma vez a cada ciclo. Esses dados são armazenados em uma variável do tipo *Array*, que tem por característica a capacidade de armazenar uma coleção de elementos identificados por um índice.

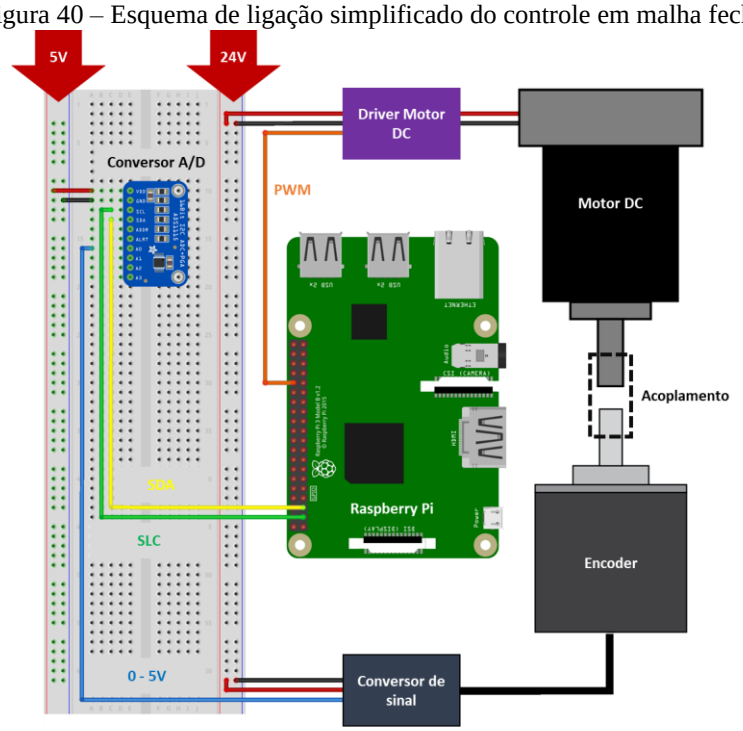
O cálculo do controlador é executado utilizando um modulo denominado DeAsync.js que transforma a execução do microserviço de assíncrona para síncrona. Isso é importante para que esse dado de saída do controlador não seja perdido durante a execução do programa. A saída do controlador é convertida para o formato JSON que é um tipo de formatação utilizada para troca de dados em JavaScript. Esse dado então é disponibilizado pelo controlador para que seja utilizado como variável manipulada (MV) do processo. Esse controlador permite ser inserido na composição de microserviços do Moleculer, funcionando através de requisições *web* e, sendo assim, possível de ser parametrizado até através de um navegador.

5.3 Aplicação Externa de Controle de Processo usando Serviços DAQ e Controle PIDPlus

Um caso de estudo de uma malha de controle de velocidade de um motor CC foi desenvolvido, buscando a validação experimental da MOA implementada para o controle de

processo. A planta a ser controlada consiste de um kit didático de controle de motor CC Motron M910 de 24V, 43W e 2200 RPM, uma fonte de alimentação 24V e 6A, um medidor de rotação do motor do tipo encoder incremental Hohnen de 600 pulsos por revolução, um circuito eletrônico de conversão de pulsos da rotação do motor em um sinal analógico (0-5 Vcc), um circuito de acionamento do motor via sinal PWM e um circuito conversor analógico/digital, como pode ser conferido pelo esquemático simplificado apresentado pela Figura 40.

Figura 40 – Esquema de ligação simplificado do controle em malha fechada



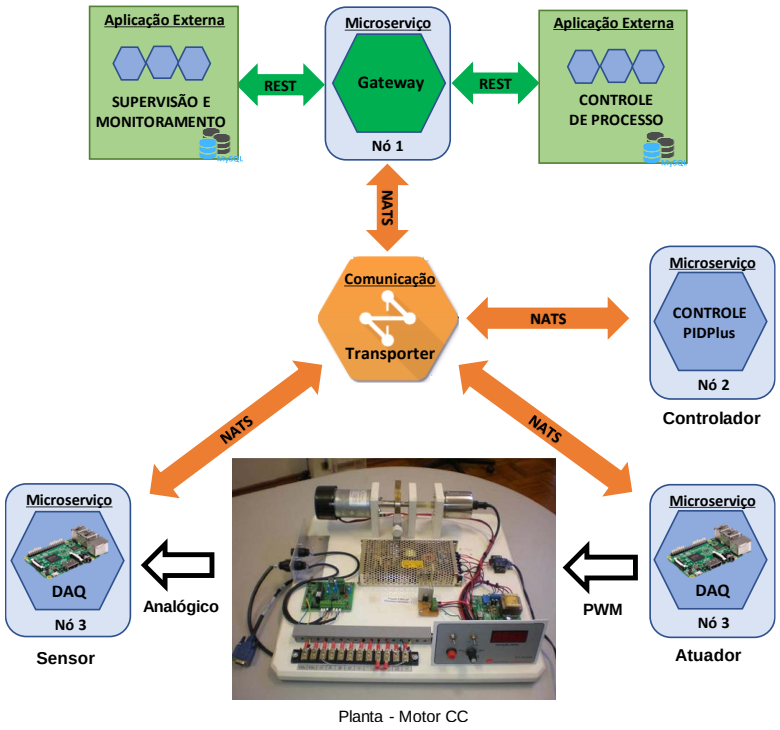
Fonte: Autor

A estrutura geral desse experimento envolvendo a MOA é apresentada na Figura 41, onde o objetivo foi verificar a capacidade de controlar e manter a malha de controle do motor estável. Nos testes realizados nenhuma carga foi aplicada ao eixo do motor. Na Figura 41, uma aplicação externa (Supervisão e Controle) foi criada para trabalhar junto à orquestração dos microserviços Controle PIDPlus, DAQ, *Transporter* e *Gateway*. Essa aplicação foi desenvolvida usando o *software* LabVIEW e é responsável pelo monitoramento (plotagem das curvas de resposta de saída), supervisão de variáveis e envio dos dados de controle do processo. O serviço *Gateway*, o serviço *Transporter* e as aplicações externas Controle e Supervisão são executados em um computador (PC).

O serviço DAQ realiza a aquisição da variável de processo (PV) e atualização da variável manipulada (MV) na planta. O serviço de Controle PIDPlus implementa o algoritmo PIDPlus em JavaScript para o cálculo do controle de processo, a partir das variáveis que são enviadas

pela aplicação externa de Supervisão e Controle. O serviço *Transporter* usado para comunicação foi o NATS (sistema de mensagens de alto desempenho que atua como uma fila de mensagens distribuídas para aplicativos). O serviço *Gateway* (Nó 1) realiza a comunicação entre a aplicação externa criada no LabVIEW com os microserviços desenvolvidos no Moleculer.

Figura 41 – Estrutura do Experimento de Controle de Processo Usando Serviços com Planta Didática.



Fonte: Autor.

O controle da malha fechada foi definido com um período de ciclo de 300 ms (valor explicado no final dessa seção). Esse ciclo foi estabelecido nesse período por meio de um loop de execução/repetição temporizado criada no próprio LabVIEW (*timed loop*), que computa a quantidade de requisições enviadas e respostas recebidas.

Por meio dessas aplicações é monitorado o comportamento do controle em malha fechada da planta didática, ou seja, o controle da velocidade do motor em relação ao ponto de ajuste (*setpoint*). O monitoramento dos dados do sensor e do atuador, coletados pelo microserviço DAQ, são disponibilizados por meio de requisições e plotados em gráficos desenvolvidos no LabVIEW. Também por meio dessas aplicações são enviados os parâmetros *Setpoint*, Kp (ganho proporcional), Ti (tempo integrativo) e Td (tempo derivativo) utilizados pelo microserviço PIDPlus.

O microserviço *Gateway* (Nó 1) executa uma conexão padronizada por meio de uma API REST entre os microserviços da arquitetura. Ele tem a função de viabilizar a comunicação dos

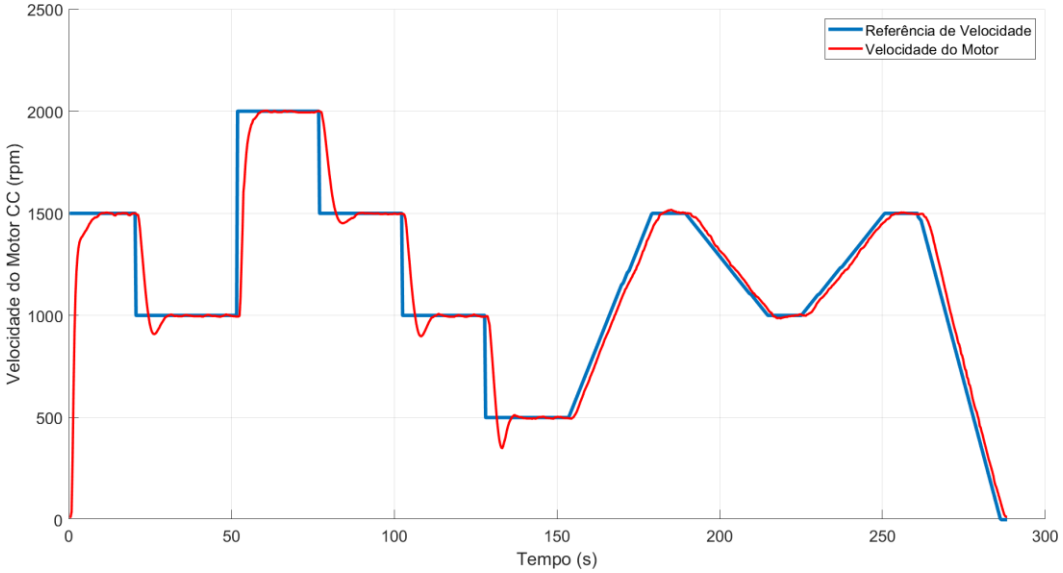
microserviços internos do Moleculer com as aplicações externas. O *Transporter* é o intermediário de envio de mensagens entre os microserviços, ou seja, a sua função é permitir a troca de dados entre os microserviços contidos na aplicação interna do Moleculer.

O microserviço de Controle PIDPlus (Nó 2) é responsável por controlar o processo, acessando a variável do processo (dados do sensor - velocidade do motor) e atuando no motor (dados do atuador - tensão aplicada ao motor), utilizando para isso o algoritmo PIDPlus desenvolvido em JavaScript. O microserviço DAQ (Nó 3) faz a aquisição de dados do sensor (variável de processo – PV) e do atuador (variável manipulada – MV), e os disponibiliza para o microserviço Controle PIDPlus.

Testes operacionais iniciais com a MOA proposta foram realizados. O experimento realizado foi o controle do motor como um serviço, mostrado na Figura 41. A Figura 42 mostra os resultados do sistema de controle de velocidade do motor e a

Figura 43 mostra o sinal de controle aplicado ao motor. Ambas as respostas foram armazenadas pelo aplicativo Supervisão. As curvas de saída (Figura 42) mostram a resposta de velocidade do motor de acordo com a referência definida. Não há requisito de desempenho de controle, pois o objetivo do experimento era avaliar: a viabilidade da realização do controle do processo usando serviços, os aspectos operacionais da MOA desenvolvida e a capacidade de controlar (rastrear a referência) e manter a estabilidade da planta. Consequentemente, sintonias diferentes do controlador resultariam em respostas diferentes (ex: respostas mais rápidas do motor).

Figura 42 – Resposta do Experimento de Controle de Processo usando Serviços.

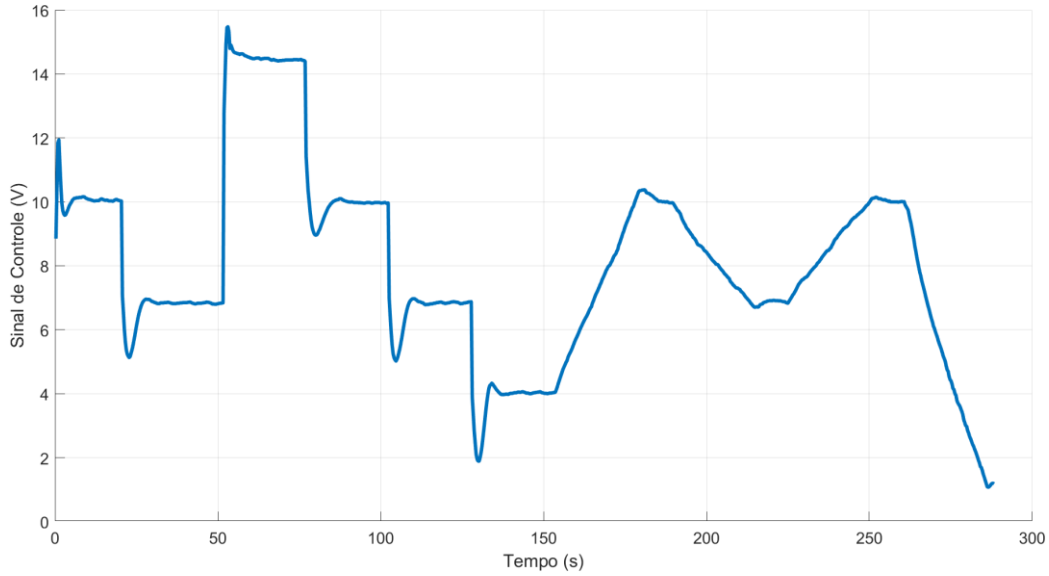


Fonte: Autor.

A Figura 42 mostra que o sistema de controle do motor é estável e controlado, pois a velocidade do motor pode rastrear o ponto de ajuste definido, considerando as entradas de degrau e rampa. A

Figura 43 mostra o sinal de controle aplicado ao motor. Não há saturação de controle e a variação é suave. Embora os resultados sejam simples e exijam mais testes e análises, foi possível verificar que a aplicação do controle como um serviço da MOA implementada é viável e demonstra um grande potencial para aplicações industriais e I4.0.

Figura 43 – Sinal de Controle do Experimento de Controle de Processo usando Serviços.



Fonte: Autor.

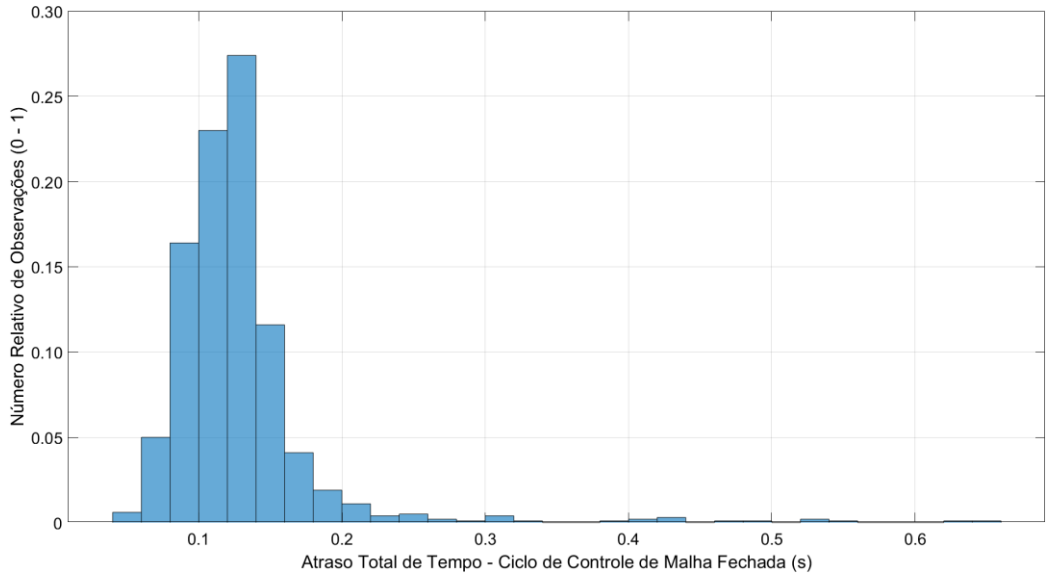
Existem várias métricas de desempenho de rede usadas para analisar sistemas de controle em rede. As principais métricas que afetam o desempenho do controle são a latência e *jitter* (LIAN; MOYNE; TILBURY, 2002). A latência é o tempo decorrido entre o envio de dados em um nó e o recebimento em outro. No caso do experimento de controle como um serviço apresentado na Figura 41, a latência seria o tempo entre uma solicitação de rede do aplicativo externo e a execução do serviço.

É difícil medir experimentalmente a latência de um serviço para receber uma solicitação e processá-la. Consequentemente, a única maneira de medir esse atraso de tempo para o experimento é a partir do tempo de ida e volta dentro de uma resposta de solicitação do aplicativo a um serviço. O tempo de ida e volta é o atraso no envio de uma solicitação pela rede da MOA e o recebimento da resposta do serviço após sua execução. Considerando o controle de malha fechada do experimento, a orquestração (ou sequência) da execução dos serviços é: serviço DAQ (sensor), serviço de controle (PIDPlus) e serviço DAQ (atuador), comunicando

via *Transporter* e *Gateway* com a aplicação externa. Somando os atrasos de tempo desses serviços, é possível obter a latência da orquestração do experimento de controle como serviço.

A Figura 44 mostra a distribuição do histograma dos tempos obtidos com a medição de ida e volta das requisições no experimento realizado. A altura de cada barra é o número relativo de observações (número de observações no compartimento / número total de observações). A soma das alturas das barras é 1. A distribuição é irregular, não se aproximando do valor médio de 128 ms e tem a presença de valores discrepantes (*outliers*), o que significa que não há determinismo para a execução do circuito fechado. Esse resultado era esperado, pois também são consideradas as comunicações de requisição e resposta do *Transporter* e *Gateway* durante os ciclos de controle. Essa variabilidade também justifica a escolha do algoritmo PIDPlus para o serviço de controle, pois ele é capaz de lidar com essas variações do tempo de execução.

Figura 44 – Histograma do Experimento de Controle usando Serviços - Tempo da Orquestração



Fonte: Autor.

Um cálculo estatístico foi feito a partir dos dados de atrasos visando a obtenção da latência média da comunicação e execução dos serviços para o experimento. A variabilidade dos valores de latência fornecerá o *jitter* (J) na MOA desenvolvida. A Tabela 1 resume os resultados de tempo dos serviços e a execução da aplicação do experimento.

Os altos valores de *jitter* na Tabela 1 confirmam a grande variação do ciclo de controle de malha fechada (aplicativo de controle de processo) verificado na Figura 44. A mesma conclusão também pode ser aplicada à latência de execução dos serviços. As comunicações do *Transporter* e *Gateway* nas respostas às solicitações de serviços são a razão da variabilidade.

Tabela 1 – Experimento de Análise de Tempo para o Controle como um Serviço

Descrição	Latência [ms]	Jitter [ms]
Serviço DAQ: Ação de Aquisição do Sensor (PV)	30,5	21,4
Serviço Controle: Ação PIDPlus	41,4	27,5
Serviço DAQ: Ação de atualização do Atuador (MV)	34,3	33,7
Ciclo de Controle de Malha Fechada	128,4	55,7

Considerando as latências de cada serviço na Tabela 1, a ação de controle é a responsável pela maior parte do tempo do ciclo de controle do circuito fechado. Entretanto, a latência das requisições do serviço DAQ, tanto para leitora quanto para escrita, em média, possui tempos aproximados.

O valor mínimo do tempo de ciclo de controle em malha fechada considera a soma dos valores de latência e *jitter* relacionados à orquestração das aplicações criadas para este experimento. Para que não houvesse perda de requisições (aquisição do sensor, computação do controle e atualização do sinal de controle na planta), um ciclo mínimo de 300 ms foi escolhido com base nos resultados estatísticos obtidos. Tomando como base o documento de Especificação Técnica para Fieldbus - ET-5290.00-2000-800-MDP-001 da Petrobras, que no item 5.2 estabelece critérios de desempenho, pode-se afirmar que o ciclo de 300 ms atende o tempo máximo de resposta para malhas de controle de processos, uma vez que, o tempo máximo especificado para variáveis de vazão, pressão e pressão diferencial é de 500 ms, e para variáveis de temperatura, nível e analítica é de 1000 ms.

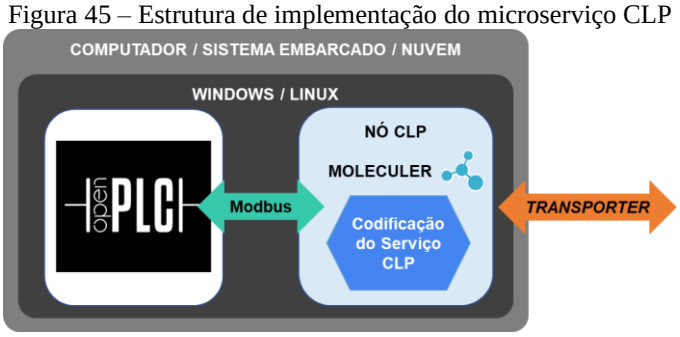
5.4 Desenvolvimento do Microserviço CLP

O microserviço CLP é responsável por integrar as funcionalidades de um CLP na arquitetura do Moleculer com o objetivo de executar lógicas e sequenciamento de tarefas que são executadas em CLPs convencionais, bem como a aquisição e o monitoramento de dados de entradas e saídas do controlador. No entanto, o diferencial dessa aplicação é que ao ser disponibilizada como um serviço, o microserviço CLP pode ser implantado em múltiplas plataformas como computador, sistema embarcado ou nuvem, conforme mostrado na Figura 45.

Para que isso fosse realizado foi necessário integrar um microserviço com o OpenPLC, que une as funcionalidades de um CLP com os benefícios de uma arquitetura aberta de *hardware* e *software*. Na estrutura do microserviço CLP implementado com o *framework* Moleculer, são disponibilizadas ações relacionadas às funcionalidades principais do serviço: programação do controlador, monitoramento de variável de entrada e monitoramento de variável de saída. As ações de monitoramento de I/O do microserviço podem ser atreladas a

diferentes tipos de I/O, como entradas e saídas analógicas e digitais. A interface entre o código do microserviço no Moleculer e as variáveis de entradas e saídas do OpenPLC é feita usando o protocolo de comunicação Modbus TCP.

Figura 45 – Estrutura de implementação do microserviço CLP



Fonte: Autor.

Da mesma forma como os demais microserviços do Moleculer, as ações são disparadas por meio de requisições HTTP direcionadas à uma rota especifica. No microserviço CLP foram criadas as seguintes ações:

- ✓ readCoils – para a leitura de saídas digitais.
- ✓ readInputs – para a leitura de entradas digitais.
- ✓ rAnalogOut – para a leitura de saídas analógicas.
- ✓ rAnalogIn – para a leitura de entradas analógicas.
- ✓ writeCoils – para a escrita de saídas digitais.
- ✓ wAnalogOutSP – para a escrita do parâmetro de *setpoint* do controlador PID.
- ✓ wAnalogOutKP – para a escrita do parâmetro de ganho proporcional do controlador PID.
- ✓ wAnalogOutTR – para a escrita do parâmetro de integração do controlador PID.
- ✓ wAnalogOutTD – para a escrita do parâmetro de derivação do controlador PID.

Para melhor entendimento, para acionar a saída digital 7 do OpenPLC por meio do microserviço CLP, seria necessário acessar a rota que dispara a ação writeCoils, onde o endereço de acesso à essa rota é o seguinte:

<http://localhost:3000/plcIO/writeCoils?address=7&value=1>

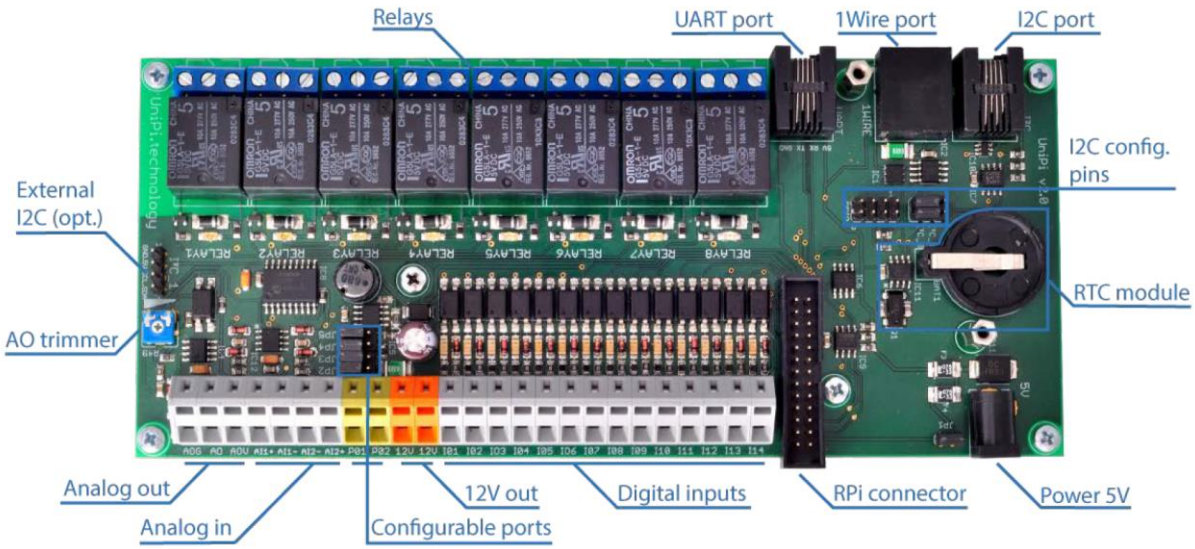
Observando o endereço descrito acima temos, logo após a definição da porta de comunicação 3000 (padrão do Moleculer), o nome do microserviço “plcIO” seguido da ação “writeCoils”. Após o ponto “?” é definido o endereço da saída do OpenPLC a ser acessada

“address=7” e o valor que essa variável irá assumir “value=1”. Portanto através dessa rota está sendo atribuído o valor de 1 para a saída 7 do OpenPLC.

Nesse trabalho o microserviço CLP foi implantado no sistema embarcado Raspberry Pi, onde o OpenPLC e o Moleculer são integrados. Além disso uma placa UniPi 1.1 é utilizada em conjunto com a Raspberry. A UniPi 1.1 é uma placa de extensão para o minicomputador Raspberry Pi, permitindo que ele funcione como um controlador lógico programável (CLP). Como mostrado na Figura 46, a UniPi possui as seguintes características:

- ✓ 8x relés de comutação (classificados para 250V AC / 10A ou 30V DC / 10A)
- ✓ 12x entradas digitais isoladas galvanicamente (tensão de 0-3,5 VDC para tensão lógica de 0 e 5-24 VDC para duração lógica mínima de pulso de 1,5 ms)
- ✓ 2 entradas analógicas de 0-10V
- ✓ 1x saída analógica de 0-10V
- ✓ 1 × porta 1-Wire para conexão a termômetros 1-Wire e outros sensores
- ✓ 1 × porta I2C para expansão
- ✓ 1 × porta UART para soquete de comunicação serial externa para bateria RTC (relógio em tempo real)

Figura 46 – Placa UniPi 1.1



Fonte: Unipi (2020)

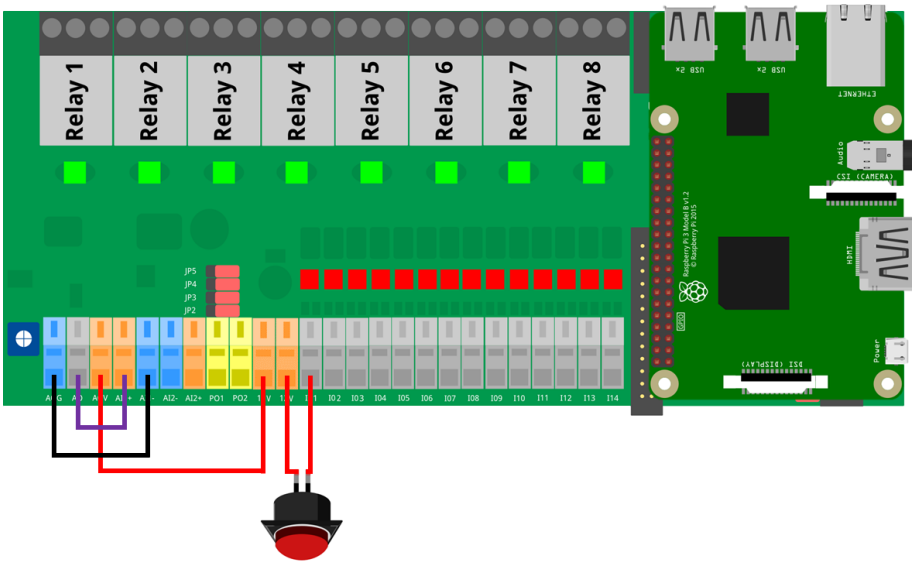
Para verificar o funcionamento do microserviço CLP foram acessadas as rotas definidas para conferir os valores das variáveis de entrada e de saída do OpenPLC para a UniPi. Para isso um programa teste foi elaborado no OpenPLC Editor e compilado para a Raspberry Pi com as seguintes funcionalidades:

- ✓ Acionar e verificar o estado logico dos relés

- ✓ Verificar o estado lógico das entradas digitais
- ✓ Estabelecer e verificar o valor da saída analógica
- ✓ Verificar o valor das entradas analógicas

O esquema de ligação utilizado para os testes pode ser conferido pela Figura 47, onde foi utilizada a própria saída analógica da UniPi para acionar sua entrada analógica. Na entrada I01 foi instalado um botão, que quando pressionado dispara o acionamento dos relés em sequência, com um intervalo de 2 segundos. Dessa forma é possível através da resposta das requisições feitas ao microserviço CLP ver o comportamento da lógica implementada.

Figura 47 – Esquema de ligação para o teste do microserviço CLP



Fonte: Autor

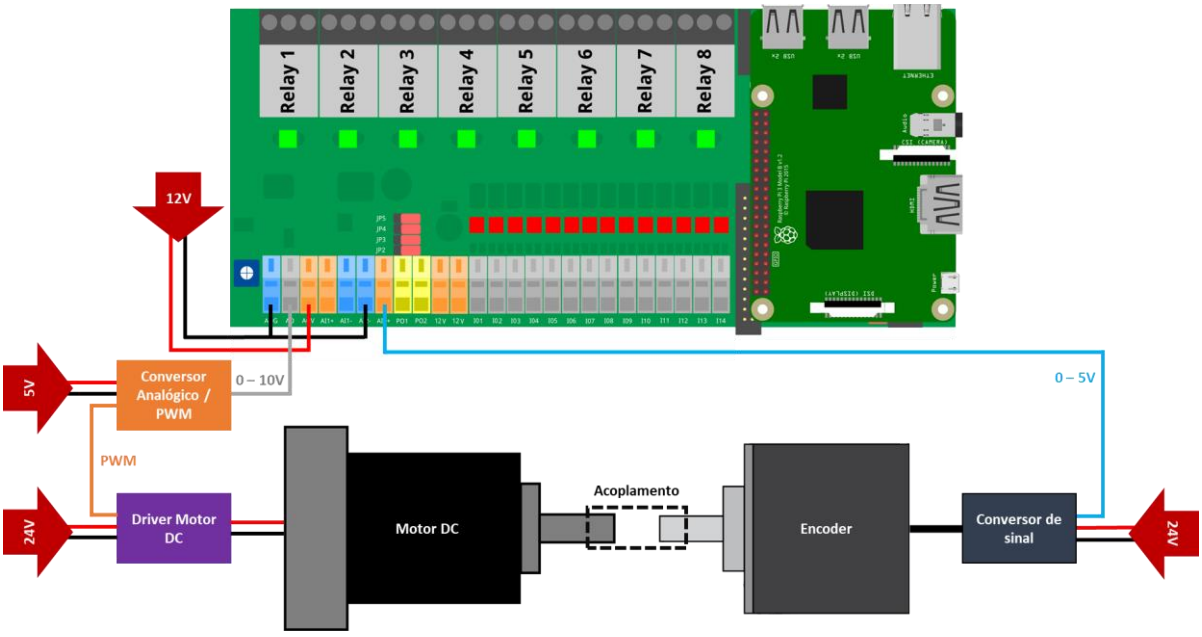
5.5 Aplicação Interna de Monitoramento, Controle e Automação de Processos usando o Microserviço CLP

A Aplicação Interna de Monitoramento tem a função de demonstrar graficamente os dados das entradas e saídas do microserviço CLP, bem como os sinais do controlador PID nativo do OpenPLC. Essa aplicação foi implementada de forma a funcionar como uma aplicação interna do Moleculer, ou seja, que se comunica com os demais microserviços através do *Transporter* (NATS). Através do comando *call* do Moleculer, se estabelece uma conexão com um determinado microserviço e uma ação com parâmetros é chamada. Isto consiste em uma configuração de comunicação interna entre os microserviços, diferentemente das aplicações externas ao Moleculer, que necessitam de um microserviço API *Gateway* para viabilizar a comunicação com os microserviços que constituem a MOA.

Para validar o funcionamento do microserviço CLP, um experimento foi elaborado para realizar o controle de um motor CC real em malha fechada em conjunto com a automação discreta de uma planta industrial virtual. A planta real é a mesma utilizada no experimento descrito na seção 5.3 (Aplicação Externa de Controle de Processo usando Serviços DAQ e Controle PIDPlus) composta de um motor CC Motron M910 de 24V e 43W, um medidor de rotação do motor do tipo encoder incremental Hohnen de 600 pulsos por revolução, um circuito eletrônico para a conversão dos pulsos gerados pela rotação do motor em um sinal analógico (0-5 Vcc) e um circuito para o acionamento do motor via sinal PWM.

A Figura 48 apresenta o esquema simplificado da planta de controle em malha fechada. Em conjunto com a planta real estão integrados o sistema embarcado Raspberry Pi e a placa de extensão UniPi 1.1. O controle do motor é feito por meio do microserviço CLP que utiliza o controlador PID do OpenPLC. A saída analógica (0 – 10Vcc) da UniPi é convertida em um sinal PWM que aciona o driver do motor CC. A rotação do motor é captada por um encoder acoplado ao seu eixo. O sinal transmitido pelo encoder é convertido para um sinal analógico (0 – 5Vcc) e conectado à entrada de leitura de sinal analógico da UniPi.

Figura 48 – Esquema simplificado da planta de controle em malha fechada



Fonte: Autor

A planta industrial virtual foi desenvolvida utilizando um simulador de fábrica 3D utilizado para aplicar tecnologias de automação denominado Factory I/O. É um *software* que permite construir rapidamente uma fábrica virtual usando uma seleção de peças industriais comuns. Geralmente o Factory I/O é utilizado como uma plataforma de treinamento de

Controlador Lógico Programável (CLP), uma vez que os CLPs são os controladores mais comuns encontrados em aplicações industriais. A Figura 49 mostra a planta virtual criada no Factory I/O para realizar o experimento com os microserviços.

Figura 49 – Planta Industrial em Simulação no *software* Factory I/O

Fonte: Autor

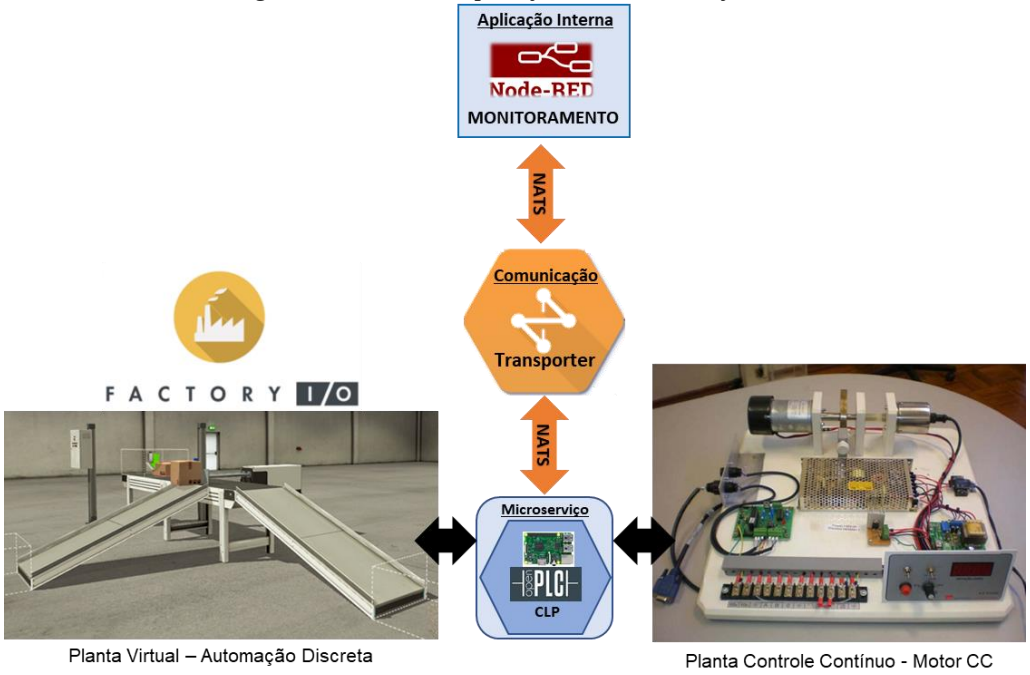
Esse cenário de simulação de planta industrial consiste em selecionar caixas de dois tamanhos diferentes, e um atuador desvia a caixa mais alta do curso da esteira para a rampa lateral. A planta virtual é composta de um painel elétrico com botoeiras de comandos de Liga, Desliga e Emergência. Um sinaleiro, também inserido no painel, indica quando o botão de Emergência está acionado. A esteira transportadora de caixas é acionada através das botoeiras contidas no painel elétrico.

Dois modelos de caixas distintos são inseridos aleatoriamente, onde a seta verde apontada para baixo está inserida. As caixas são selecionadas pelos sensores instalados ao lado da esteira. Quando os dois sensores são acionados ao mesmo tempo, a lógica construída no OpenPLC habilita o atuador após um determinado tempo, para que a caixa alta seja desviada do seu curso pela rampa lateral. Caso somente o sensor inferior seja acionado, o atuador não altera seu estado, permitindo que as caixas baixas prossigam por todo o curso da esteira, deslizando pela rampa localizada na extremidade da esteira. As setas vermelhas apontadas para cima removem da cena as caixas que deslizam pelas rampas.

Essa aplicação de controle de processo e automação consiste de uma arquitetura de microserviços distribuída do Moleculer, como é apresentado na Figura 50. Trata-se de uma composição de serviços por orquestração, pois os microserviços trabalham com o mecanismo

de requisição/resposta, ou seja, a Aplicação Interna de Monitoramento envia uma requisição ao serviço CLP, que por sua vez retorna a resposta à aplicação de Monitoramento.

Figura 50 – MOA da aplicação do microserviço CLP



Fonte: Autor

Na aplicação da Figura 50, o Factory I/O, *software* que simula a planta industrial, se comunica via protocolo Modbus TCP diretamente com o OpenPLC, que é responsável por controlar o funcionamento da planta industrial. Toda a lógica de controle do CLP foi desenvolvida em *Ladder* por meio do OpenPLC Editor.

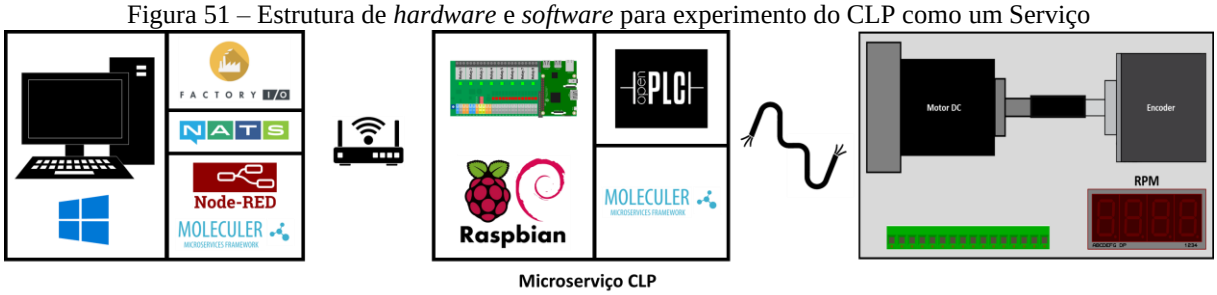
O microserviço CLP foi implantado em um sistema embarcado (Nó 1). Dessa forma, é possível comunicar seus dados de interesse através do *Transporter* NATS para a aplicação de Monitoramento (Nó 2), implantada num computador usando o Node-RED. No caso em questão, os dados de interesse do microserviço CLP são os estados das variáveis de entrada e saída e do controlador PID do OpenPLC. Essa aplicação permite monitorar o comportamento do CLP em tempo real, a partir de qualquer dispositivo que contenha acesso à um navegador *web* e à infraestrutura de rede em que os dispositivos de interesse estejam conectados.

Para criar essa Aplicação de Monitoramento com maior facilidade foi utilizado o Node-RED, pois ele possui blocos de funções do *framework* Moleculer (como apresentado no Anexo 0) e também blocos específicos para a construção de *dashboard*, o que facilita o desenvolvimento de um sistema de supervisão baseado em uma interface gráfica. O Node-RED é uma ferramenta de programação para conectar dispositivos de *hardware*, Interface de Programação de Aplicativos (*API - Application Programming Interface*) e serviços online. Ele

fornece um editor baseado em um navegador *web* que utiliza uma abordagem de programação gráfica.

A concepção desse experimento foi possível através da integração de algumas tecnologias. A Figura 51 demonstra os dispositivos e *software* utilizados, bem como a estrutura à qual foram aplicados. São utilizados um computador (PC), um sistema embarcado Raspberry Pi, uma placa de extensão UniPi 1.1 e uma planta de controle em malha fechada.

Figura 51 – Estrutura de *hardware* e *software* para experimento do CLP como um Serviço



Fonte: Autor

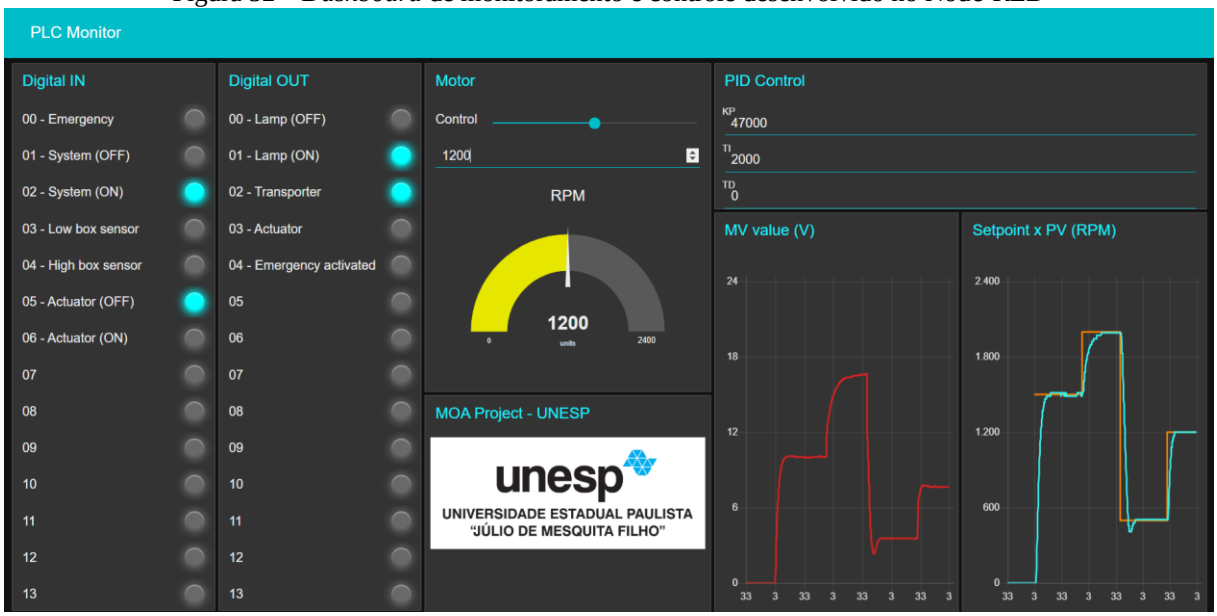
No PC está instalado o sistema operacional Windows. Nesse sistema operacional estão instalados o *software* Factory I/O, o *transporter* NATS e o Node-RED. No Node-RED foi desenvolvida a Aplicação Interna de Monitoramento, responsável por requisitar e disponibilizar os dados do microserviço CLP através de uma interface de visualização criada. No sistema embarcado está instalado o sistema operacional Raspbian (Linux), que contém o *software* OpenPLC e o microserviço CLP. O OpenPLC foi instalado em uma placa Raspberry Pi que está conectada à placa de expansão Unipi 1.1. A planta de controle contém um motor CC para regulação da velocidade de rotação.

A Figura 52 apresenta a interface de monitoramento dessa aplicação. Do lado esquerdo são apresentados os estados lógicos das entradas e saídas digitais do microserviço CLP, que controlam a planta virtual construída no Factory I/O. Na parte central está o controle de *setpoint* do motor em rotações por minuto. No lado superior direito estão os campos destinados ao envio dos parâmetros do controlador PID (KP, TI e TD). Na parte inferior direita existem dois gráficos que mostram o comportamento da saída de tensão (V) aplicada ao motor (variável manipulada), e a velocidade do motor (variável de processo) em relação ao *setpoint* (RPM) respectivamente.

O experimento realizado com o microserviço CLP apresentou confiabilidade e desempenho compatíveis para a aplicação de automação virtual e controle de processo real, não apresentando erros operacionais ou falhas de comunicação. Foi possível verificar que a arquitetura de microserviços tem a capacidade de controlar e monitorar tanto a automação de

uma planta industrial virtual quanto o controle em malha fechada da velocidade de um motor CC de uma planta real, e ao mesmo tempo.

Figura 52 – *Dashboard* de monitoramento e controle desenvolvido no Node-RED



Fonte: Autor

Os resultados são satisfatórios em ambos os casos, uma vez que, embora a estrutura da planta virtual seja relativamente simples, o sistema como um todo opera de forma eficaz. Já para a planta de controle real, o fato de o motor rastrear a velocidade estabelecida pelo ponto de ajuste, como mostra a Figura 52, mostra a efetividade do sistema em relação ao controle do processo. Outro ponto a ser destacado é a flexibilidade operacional do sistema, que pode ser acessado a partir de navegadores *web*, que são comumente utilizados em diferentes tipos de dispositivos, tais como, computadores e *smartphones*. O acesso operacional por meio de dispositivos móveis também oferece mobilidade ao usuário do sistema, se apresentando como uma vantagem em relação às IHMs que são fixas em seus equipamentos, na maioria das aplicações industriais.

5.6 Discussões e Considerações Finais

A integração das tecnologias de automação e informática industrial numa arquitetura orientada a microserviços (MOA) permitiu o uso e compartilhamento de microserviços para obtenção de uma arquitetura industrial flexível, escalável, interoperável, distribuída em rede. A implementação da Automação Industrial como um Serviço desenvolvida neste trabalho representa uma mudança de paradigma nas interações entre os diferentes sistemas industriais,

equipamentos, aplicações e usuários, a qual vem de encontro com as mudanças e requisitos das aplicações da Indústria 4.0.

É importante discutir os benefícios e as desvantagens da MOA desenvolvida considerando o experimento realizado neste trabalho. A principal desvantagem da solução proposta é obviamente a simplicidade dos experimentos realizados. Um outro ponto a ser destacado é a complexidade de gerenciamento em grandes aplicações contendo muitos serviços. Por ser um paradigma diferente das arquiteturas monolíticas industriais tradicionais em termos de desenvolvimento, manutenção e implantação, o aprendizado de novos conceitos e a utilização de novos ambientes de desenvolvimento ainda podem ser considerados como desvantagens dessa arquitetura para aplicações industriais.

Apesar de existirem desvantagens, as vantagens de adoção e uso das arquiteturas de microserviços em aplicações de automação industrial são maiores e mais relevantes. O principal benefício dessas arquiteturas é a integração vertical entre camadas e a interoperabilidade de todo o sistema. Esse benefício é alcançado através da comunicação transparente e padronizada entre serviços e aplicações realizada pelo serviço de comunicação (*Transporter*). Embora o benefício dessa integração não tenha sido significativo nos experimentos deste trabalho por conta da simplicidade, a facilidade de monitoramento ficou clara devido à interoperabilidade e acesso padronizado às informações dos serviços e aplicações.

A adoção do *framework* Moleculer como base para a MOA desenvolvida foi um diferencial e trouxe algumas vantagens como evitar o desenvolvimento necessário de toda a infraestrutura dos microserviços, como o formato de dados de serviço e o barramento de serviço. Outro diferencial da MOA desenvolvida é a flexibilidade para composição de serviços por Orquestração e Coreografia, ainda que neste trabalho somente tenha sido usado a primeira.

Além de fornecer uma infraestrutura padronizada de microserviços à MOA, o Moleculer adicionou outras funcionalidades, como monitoramento de serviço, API padrão para comunicações externas (*Gateway*) e comunicação entre serviços de forma transparente (*Transporter*). Em particular, o monitoramento de serviços, como integridade e status de falha, é importante. O Moleculer inclui um *Circuit Breaker* e um *Balanced Events*. Resumidamente, o *Circuit Breaker* verifica o *status* do serviço e considera as solicitações de baixo desempenho ou malsucedidas para definir o *status* como íntegro ou interrompido. O *Balanced Events* distribui as solicitações em um conjunto de serviços iguais. O *Circuit Breaker* permite que o *Balanced Events* solicite apenas serviços que estão em bom estado de funcionamento. Isso reduz o tráfego de dados na MOA e diminui a latência devido à retransmissão.

A manutenção (alteração/atualização) e o comissionamento (execução/ replicação) da arquitetura de microserviços, elaborada com o Moleculer, podem ser feitos online e

remotamente através da rede. O desenvolvimento de cada microserviço em um nó facilita a replicação e reuso desses serviços em outras aplicações. A redundância de serviços, ainda que não contemplada nos experimentos realizados, também representa outra importante característica da arquitetura de microserviços para aplicações de automação e controle. A redundância de serviços pode ser implementada através da replicação (cópia) do mesmo serviço. Nesse caso, o serviço de comunicação do Moleculer automaticamente verifica que existem duas ou mais instâncias (redundantes) do mesmo serviço, e em caso em requisição desse serviço, ele automaticamente seleciona a melhor dessas instâncias para execução naquele momento.

6 CONCLUSÕES

O uso de arquiteturas orientadas a serviços em aplicações industriais relacionadas à Industria 4.0 é uma realidade necessária para a promoção da interoperabilidade e interação vertical de dispositivos e sistemas, além da habilitação de uma arquitetura distribuída e modular.

Este trabalhou apresentou conceitos sobre sistemas que integram as arquiteturas orientadas a serviços (SOA e MOA), bem como suas aplicações e importância na estrutura da Indústria 4.0. O uso de arquiteturas orientadas a serviços empregadas para o uso industrial tem se consolidado, despertando grande interesse em pesquisas e desenvolvimentos recentes, tendo como foco principal a promoção da interoperabilidade e interação vertical de dispositivos e sistemas, além do uso de dispositivos de forma distribuída e modular dentro de um ambiente industrial.

As funcionalidades e os benefícios de uma arquitetura industrial orientada a microserviços (MOA) voltada para aplicações no contexto da I4.0 foram discutidas. Detalhes operacionais da arquitetura e da estrutura utilizada, bem como a comunicação e composição dos serviços foram apresentados. O desenvolvimento dos microserviços e aplicações propostas neste trabalho foram apresentados, sendo que o ponto chave foi a utilização do *framework* Moleculer, que simplifica a implantação de uma arquitetura orientada a microserviços, uma vez que toda a infraestrutura computacional de SOA não precisou ser desenvolvida, como visto em trabalhos na literatura, reduzindo o tempo de desenvolvimento da arquitetura e permitindo que o foco fosse dado ao desenvolvimento dos serviços e aplicações voltadas para automação e controle.

Este trabalho discutiu a criação e uso dos microserviços DAQ, CLP e Controle PIDPlus e de aplicações de Controle e Monitoramento via orquestração dos serviços. Considerando O microserviço CLP, a possibilidade de se ter um controlador programável como um serviço em nuvem é indicada para aplicações em que não se exige determinismo e é vantajosa por não ser necessária a implantação de controladores físicos na planta. O microserviço DAQ se assemelha à operação de uma remota de I/O em rede, com a vantagem de ser programável e poder incorporar funcionalidades customizadas pra cada aplicação. O microserviço de Controle flexibiliza o desenvolvimento e aplicação de algoritmos de controle conforme a necessidade da aplicação. Nesse trabalho, o uso do algoritmo de controle PIDPlus, usado em controle via rede, se mostrou adequado para controlar as malhas de processo que se comunicam em rede através do serviço de comunicação (*transporter*). Além disso, todos os microserviços apresentaram tempo de resposta compatíveis com as aplicações de controle de processos e automação.

Dois experimentos foram realizados buscando a validação da arquitetura proposta para aplicações industriais, onde um apresentou o controle de velocidade de um motor CC implementado como serviço. Para este experimento, vários microserviços foram utilizados (DAQ, Controle, *Transporter* e *Gateway*). Os resultados operacionais demonstraram a viabilidade e o potencial das arquiteturas baseadas em serviços para suportar aplicações de automação e controle de processos no contexto da I4.0. A arquitetura MOA usada na implementação da malha de controle como um serviço provou ser vantajosa no sentido de prover desacoplamento entre os níveis de dispositivo e aplicação, além de promover interoperabilidade e interação vertical para a solução de controle de processos.

O outro experimento envolveu os microserviços CLP, *Transporter* e a aplicação de Monitoramento em aplicações de controle de processo e automação. Nesse experimento uma planta industrial virtual foi controlada e monitorada, onde foi possível verificar em tempo real a atualização do estado logico das variáveis de entradas e saídas do CLP. Também foi realizado o controle de velocidade de um motor CC utilizando o PID nativo do OpenPLC em conjunto com os microserviços, onde a resposta de velocidade do motor em relação ao ponto de ajuste, bem como o sinal da variável manipulada puderam ser monitorados através do *Dashboard*. Além do monitoramento, por meio do *Dashboard* foi possível também enviar os parâmetros para o controlador e definir o ponto de ajuste de velocidade do motor.

As aplicações apresentadas nesse trabalho mostram que a Arquitetura Orientada a Microserviços é promissora quanto ao desenvolvimento de soluções para a I4.0. Temas relacionados à segurança e redundância, que não foram abordadas nesse trabalho, oferecem possibilidades para pesquisas futuras.

REFERÊNCIAS

BANGEMANN, T. et al. State of the art in industrial automation. Industrial cloud-based cyber-physical systems. *In*: COLOMBO A. W. et al**. Industrial cloud-based cyber-physical systems**. Cham: Springer, 2014. p. 23-47. DOI: 10.1007/978-3-319-05624-1_2.

BASSI, L. Industry 4.0: Hope, hype or revolution?. *In*: INTERNATIONAL FORUM ON RESEARCH AND TECHNOLOGIES FOR SOCIETY AND INDUSTRY (RTSI), 3., 2017, Modena. **Proceedings [...]**. Piscataway: IEEE, 2017. p. 1-6. DOI: 10.1109/RTSI.2017.8065927. Disponível em: https://ieeexplore.ieee.org/abstract/document/8065927. Acesso em: 08 fev. 2019.

BEREMIZ. Disponível em: <http://www.beremiz.org/>. Acesso em: Janeiro, 2019.

BIGHETI J. A.; FERNANDES M. M.; RISSO S. L.; ARÃO D. P.; GODOY E. P. Controle de Processos baseado em Serviços para a Indústria 4.0. *In*: CONGRESSO BRASILEIRO DE INSTRUMENTAÇÃO, SISTEMAS E AUTOMAÇÃO, 2019, Campinas. **Anais** [...] Campinas: GALOÁ, 2019. Disponível em: https://proceedings.science/cobisa-2019/papers/controle-de-processos-baseado-em-servicos-para-a-industria-4-0. Acesso em: 11 jul. 2019.

BIGHETI, J. A.; FERNANDES, M. M.; GODOY, E. P. Control as a service: a microservice approach to industry 4.0. *In*: WORKSHOP ON METROLOGY FOR INDUSTRY 4.0 AND IOT, 2., 2019, Naples. **Proceedings** [...]. Piscataway: IEEE, 2019. p. 438-443. DOI: 10.1109/METROI4.2019.8792918. Disponível em: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8792918. Acesso em: 12 ago. 2019.

BIGHETI, J. A.; RISSO, S. L.; CALDIERI, M. R.; GODOY, E. P. Proposta de arquitetura orientada a microserviços para aplicações de internet das coisas industrial. *In*: CONGRESSO BRASILEIRO DE AUTOMÁTICA, 22, 2018, João Pessoa, **Anais** […]. João Pessoa: SWGE, 2018. Disponível em: https://plataforma.swge.com.br/PROCEEDINGS/PDF/CBA2018-0971.pdf. Acesso em: 20 jan. 2019.

BLANCHET, M. et al. **Industry 4.0 The new industrial revolution.** Munich: Roland Berger, 2014. Disponível em: http://www.iberglobal.com/files/Roland_Berger_Industry.pdf. Acesso em: 08 fev. 2019.

BORANGIU, T.; TRENTESAUX, D.; THOMAS, A.; LEITÃO, P.; BARATA, J. Digital transformation of manufacturing through cloud services and resource virtualization. **Computers in Industry**, Amsterdam, v. 108, p. 150-162, Jun. 2019. DOI: 10.1016/j.compind.2019.01.006. Disponível em: http://www.sciencedirect.com/science/article/pii/S0166361519300107. Acesso em: 09 set. 2019.

BUTZIN, B.; GOLATOWSKI, F.; TIMMERMANN, D. Microservices approach for the internet of things. *In*: INTERNATIONAL CONFERENCE ON EMERGING TECHNOLOGIES AND FACTORY AUTOMATION, 21., 2016, Berlin. **Proceedings** [...]. Piscataway: IEEE, 2016. Disponível em: https://ieeexplore.ieee.org/document/7733707. Acesso em: 10 out. 2018.

CALDIERI, M. R.; BIGHETI, J. A.; GODOY, E. P. Implementação do Modbus para aplicação em sistema de controle via rede sem fio. 2016. *In:* CONGRESSO BRASILEIRO DE AUTOMÁTICA, 21., 2016, Vitória. **Anais** [...]. Vitória, [*s. n.*], 2016 p.219-224. Disponível em: https://ssl4799.websiteseguro.com/swge5/PROCEEDINGS/PDF/CBA2016-0070.pdf. Acesso em: 13 mai. 2019.

CARLSSON, O.; DELSING, J.; ARRIGUCCI, F.; COLOMBO, A.W.; BANGEMANN, T.; NAPPEY, P. Migration of industrial process control systems to service-oriented architectures. **International Journal of Computer Integrated Manufacturing**, Vol. 31, No. 2, pp. 175-198, 2018.

CHEN, Y.; DU, Z.; GARCÍA-ACOSTA, M. Robot as a service in cloud computing. Proceedings. IEEE INTERNATIONAL SYMPOSIUM ON SERVICE-ORIENTED SYSTEM ENGINEERING, SOSE, 5., 2010. **Proceedings** [...]. Piscataway: IEEE, 2010. p. 151–158. Disponível em: https://ieeexplore.ieee.org/document/5570010. Acesso em: 14 mai. 2019.

CIAVOTTA, M.; ALGE, M.; MENATO, S.; ROVERE, D.; PEDRAZZOLI, P. A Microservice-based Middleware for the Digital Factory. Procedia Manufacturing, Vol. 11, 2017, pp. 931-938.

COLOMBO, A. W. et al. Industrial cyberphysical systems: a backbone of the fourth industrial revolution. **IEEE Industrial Electronics Magazine**, Piscataway, v. 11, n. 1, p. 6–16, 2017. Disponível em: https://ieeexplore.ieee.org/abstract/document/7883993. Acesso em: 10 out. 2018.

COLOMBO, A. W.; KARNOUSKOS, S.; BANGEMANN, T. Towards the next generation of industrial Cyber-Physical Systems. *In:* COLOMBO, A. W. et al. (ed.). **Industrial Cloud-Based Cyber-Physical Systems**. Cham: Springer International Pub, 2014. p. 1–22.

DELSING, J. et al. A migration approach towards a SOA-based next generation process control and monitoring. *In*: ANNUAL CONFERENCE OF THE IEEE INDUSTRIAL ELECTRONICS SOCIETY IECON, 37., 2011, Melbourne. **Proceedings** [...]. Piscataway, IEEE, 2011. p. 4472–4477. Disponível em: https://ieeexplore.ieee.org/document/6120045 Acesso em: 10 out. 2018.

DUSTDAR, S.; PAPAZOGLOU, M. P. Services and Service Composition – An introduction (Services und Service Komposition – Eine Einführung). **it - Information Technology**, Muenchen, v. 50, n. 2, p. 86–92, 2008. Disponível em: https://pdfs.semanticscholar.org/422b/e74a233ca10c0fd93e453da06cda22fed7a8.pdf . Acesso em: 15 out. 2018.

ESPÍ-BELTRAN, J.V.; GILART-IGLESIAS, V.; RUIZ-FERNANDEZ, D. **Enabling distributed manufacturing resources through SOA**: The REST approach. Robotics and Computer-Integrated Manufacturing, Vol. 46, 2017, pp. 156-165.

GIRBEA, A.; SUCIU, C.; NECHIFOR, S.; SISAK, F. Design and Implementation of a Service-Oriented Architecture for the Optimization of Industrial Applications. **IEEE Transactions on Industrial Informatics**, vol. 10, no. 1, pp. 185-196, Feb. 2014.

HEGAZY, T.; HEFEEDA, M. Industrial automation as a cloud service. **IEEE Transactions on Parallel and Distributed Systems**, Piscataway, v. 26, n. 10, p. 2750–2763, 2015. Disponível em: https://ieeexplore.ieee.org/document/6908023. Acesso em: 22 out. 2018.

IMC-AESOP Project. **IMC-AESOP Project**, 2011. Disponível em: http://www.imcaesop.eu. Acesso em: Julho, 2019.

INNERBICHLER, I.; GONUL, S.; DAMJANOVIC-BEHRENDT, V.; MANDLER, B.; STROHMEIER, F. NIMBLE collaborative platform: Microservice architectural approach to federated IoT. Global Internet of Things Summit (GIoTS), Geneva, 2017, pp. 1-6.

JAMMES, F. et al. Promising technologies for SOA-based industrial automation systems. *In:* COLOMBO, A. W. et al. (ed.). **Industrial Cloud-Based Cyber-Physical Systems**. Cham: Springer International Pub, 2014. p. 89–109.

JAZDI, N. Cyber physical systems in the context of Industry 4.0. *In:* IEEE INTERNATIONAL CONFERENCE ON AUTOMATION, QUALITY AND TESTING, ROBOTICS, AQTR, 2014, Cluj-Napoca. **Proceedings** [...]. Piscataway: IEEE, 2014. p. 1–4. Disponível em: https://ieeexplore.ieee.org/abstract/document/6857843. Acesso em: 24 nov. 2018

KAGERMANN, H.; WAHLSTER, W.; HELBIG, J. **Recommendations for implementing the strategic initiative industrie 4.0**. Frankfurt: Acatech, 2013.

KARNOUSKOS, S.; COLOMBO, A.W.; BANGEMANN, T.; MANNINEN, K.; CAMP, R.; TILLY, M.; SIKORA, M.; JAMMES, F.; DELSING, J.; ELIASSON, J.; NAPPEY, P.; HU, J.; GRAF, M. The IMC-AESOP **Architecture for Cloud-Based Industrial Cyber-Physical Systems**. In: Colombo A. et al. (eds) Industrial Cloud-Based Cyber-Physical Systems. Springer, Cham, 2014.

LEE, J.; BAGHERI, B.; KAO, H. A. A cyber-physical systems architecture for Industry 4.0-based manufacturing systems. **Manufacturing Letters**, v. 3, p. 18–23, 2015.

LEITÃO, P.; COLOMBO, A.W.; KARNOUSKOS, S. **Industrial automation based on cyber-physical system Technologies:** Prototype implementations and challenges. Computers in Industry. Vol. 81, pp. 11-25, 2016.

LEWIS, J.; FOWLER, M. **Microservices**: a definition of this new architectural term. Chicago: martinFowler.com, 2014. Disponível em: https://martinfowler.com/articles/microservices.html. Acesso em: 20 jan. 2019

LIAN, F. L.; MOYNE, J.; TILBURY, D. Network design consideration for distributed control systems. **IEEE Transactions on Control Systems Technology**, v. 10, n. 2, p. 297, 2002

LU, Y. Industry 4.0: A survey on technologies, applications and open research issues. **Journal of Industrial Information Integration**, v. 6, p. 1–10, 2017.

MOLECULER. **What is moleculer?**. [*S. l.*], c2016-2019. Disponível em: https://moleculer.services/docs/0.13/. Acesso em: 5 abr. 2018.

MORAES, E.C. **Desenvolvimento de padrões de interfaces para integração de sistemas heterogêneos na manufatura**. 247 f. Tese de Doutorado, Engenharia Industrial, Universidade Federal da Bahia, Salvador, 2017.

MOREIRA, P. F. M.; BEDER, D. M. Desenvolvimento de aplicações e microserviços: um estudo de caso. **Revista T.I.S.**, v. 4, n. 3, p. 209–215, 2016.

NEWMAN, S. **Building Microservices**. Sebastopol: O'Reilly Media, 2015.

NIKOLAIDIS, P.; DIDIC, A.; MUBEEN, S.; PEI-BREIVOLD, H.; SANDSTRÖM, K.; BEHNAM, M. Applying mitigation mechanisms for cloud-based controllers in industrial IoT applications. *In*: Internet-of-Things Symposium IoT Symposium, 15, 2015, Amsterdam, **Proceedings** [...]. Amsterdam: [s. n], 2015. Disponível em: http://www.es.mdh.se/pdf_publications/4023.pdf. Acesso em: 8 ago. 2018.

NPM. **PM2**, 2020. Disponível em: https://www.npmjs.com/package/pm2. Acesso em: 19 jul. 2020.

OLLINGER, L.; ABDO, A.; ZÜHLKE, D.; HEUTGER, H. SOA-PLC – Dynamic Generation and Deployment of Web Services on a Programmable Logic Controller. *In*: IFAC World Congress, 19, 2014, Cape Town, **Proceedings** […]. Cape Town: Elsevier Ltd, 2014. p 2622-2627, DOI: 10.3182/20140824-6-ZA-1003.02189. Disponível em: http://www.sciencedirect.com/science/article/pii/S1474667016420057. Acesso em: 10 out. 2018.

OPEN PLC. 2019. Disponível em: https://www.openplcproject.com. Acesso em: 5 maio 2019.

ORDANINI, A.; PASINI, P. Service co-production and value co-creation: The case for a service-oriented architecture (SOA). **European Management Journal**, v. 26, n. 5, p. 289–297, 2008. DOI: 10.1016/j.emj.2008.04.005. Disponível em: http://www.sciencedirect.com/science/article/pii/S0263237308000431. Acesso em: 16 ago. 2018.

PISCHING, M. A. **Arquitetura para descoberta de equipamentos em processos de manufatura com foco na Indústria 4.0**. 2018. 189 f. Tese (Doutorado) - Curso de Ciências, Escola Politécnica da Universidade de São Paulo, São Paulo, 2018. Disponível em: https://www.teses.usp.br/teses/disponiveis/3/3152/tde-05032018-133720/publico/MarcosAndrePischingCorr18.pdf. Acesso em: 20 jan. 2019.

RAJKUMAR, R.; LEE, I.; L.; SHA, L.; STANKOVIC, J. Cyber-physical systems: The next computing revolution. *In*: DESIGN AUTOMATION CONFERENCE, 2010, Anaheim, **Proceedings** […]. Anaheim: IEEE, 2014. p. 731-736. DOI: 10.1145/1837274.1837461. Disponível em: https://ieeexplore.ieee.org/document/5523280. Acesso em: 22 fev. 2019.

RICHARDS, M. **Microservices vs. service- oriented architecture**. Sebastopol: O'Reilly Media, 2016.

SAUTER, Thilo et al. The Evolution of Factory and Building Automation. **IEEE Industrial Electronics Magazine**, Piscataway, v. 5, n. 3, p.35-48, set. 2011. DOI: 10.1109/mie.2011.942175. Disponível em: https://ieeexplore.ieee.org/abstract/document/6042563. Acesso em: 22 mai. 2019.

SHENG, Quan Z. et al. Web services composition: A decade's overview. **Information Sciences**, Amsterdam: Elsevier BV, v. 280, p .218-238, Out. 2014. DOI: http://dx.doi.org/10.1016/j.ins.2014.04.054. Disponível em: https://www.sciencedirect.com/science/article/pii/S0020025514005428. Acesso em: 22 mai. 2019.

SISINNI, Emiliano et al. Industrial internet of things: challenges, opportunities, and directions. **IEEE Transactions on Industrial Informatics**, Piscataway, v. 14, n. 11, p. 4724-4734, 2018. DOI: http://dx.doi.org/10.1109/tii.2018.2852491. Disponível em: https://ieeexplore.ieee.org/document/8401919 Acesso em: 16 ago. 2018.

SONG, J. et al. Improving PID control with unreliable communications. *In*: ISA EXPO, 2006, Houston, **Proceedings** [...]. Durham: ISA, 2006. p. 105–116. Disponível em: https://www.automation.com/pdf_articles/10_improving_pid.pdf.Acesso em: 22 abr. 2019

SOUIT, S.; FATTORI, C. C.; JUNQUEIRA, F.; D. J. S. FILHO, D. J. S.; MIYAGI, P, E. Orchestrating dispersed productive systems. CONFERENCE ON EMERGING TECHNOLOGIES & FACTORY AUTOMATION (ETFA), 18., 2013, Cagliari, **Proceedings** […]. Cagliari: IEEE, 2013. p. 1-4. DOI: 10.1109/ETFA.2013.6648132. Disponível em: https://ieeexplore.ieee.org/abstract/document/6648132. Acesso em: 22 abr. 2019.

STANKOVIC, J. A. Research directions for the internet of things. **IEEE Internet of Things Journal**, Piscataway, v. 1, n. 1, p. 3–9, 2014. DOI: 10.1109/JIOT.2014.2312291. Disponível em: https://ieeexplore.ieee.org/document/6774858. Acesso em: 05 jun. 2019.

THEORIN, A.; HAGSUND, J.; JOHNSSON, C. Service orchestration with OPC UA in a graphical control language. *In:* **IEEE EMERGING TECHNOLOGY AND FACTORY AUTOMATION (ETFA)**, 2014, Barcelona. **Proceedings** [...]. Piscataway: IEEE, 2014. p.1-

6. DOI: 10.1109/etfa.2014.7005351. Disponível em: https://ieeexplore.ieee.org/abstract/document/7005351. Acesso em: 17 mai. 2019.

UNIPI. **UniPi Technology**. Disponível em: <https://www.unipi.technology/>. Acesso em: 12 out. 2020.

WOLLSCHLAEGER, M.; SAUTER, T.; JASPERNEITE, J. The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0. **IEEE Industrial Electronics Magazine**, [s.l.], v. 11, n. 1, p.17-27, mar. 2017. Institute of Electrical and Electronics Engineers (IEEE). DOI: 10.1109/mie.2017.2649104.

WU, D.; GREER, M. J.; ROSEN, D. W.; SCHAEFER, D. Cloud manufacturing: Strategic vision and state-of-the-art. **Journal Of Manufacturing Systems**, [s.l.], v. 32, n. 4, p.564-579, out. 2013. Elsevier BV. DOI: 10.1016/j.jmsy.2013.04.008.

XIAO, Z.; WIJEGUNARATNE, I.; QIANG, X. Reflections on SOA and microservices. INTERNATIONAL CONFERENCE ON ENTERPRISE SYSTEMS: ADVANCES IN ENTERPRISE SYSTEMS, 4., 2016, Melbourne, **Proceedings** […]. Melbourne: IEEE, 2016 p. 60–67. DOI: 10.1109/ES.2016.14.

ZUEHLKE, D. SmartFactory: towards a factory-of-things. **Annual Reviews In Control**, Amsterdam, v. 34, n. 1, p. 129-138, Abr. 2010. DOI: 10.1016/j.arcontrol.2010.02.008. Disponível em: https://www.sciencedirect.com/science/article/pii/S1367578810000143. Acesso em: 17 mai. 2019.

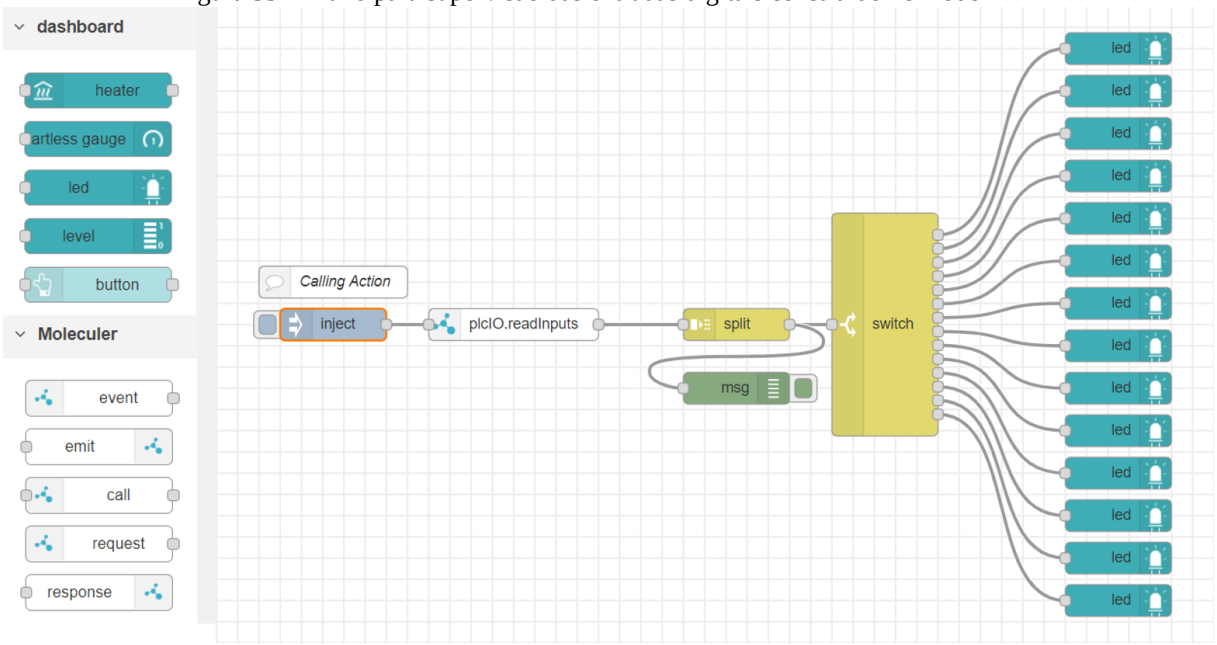
APÊNDICES

Criação de uma Aplicação Interna no Node-RED

Três bibliotecas foram instaladas no Node-RED para a construção da Aplicação de Monitoramento. A primeira contém ferramentas especificas para a construção de *dashboard* denominada *node-red-dashboard*. A segunda contém blocos com funções do *framework* Moleculer denominada *node-red-contrib-moleculer*. A terceira é um complemento para o desenvolvimento do *dashboard*, denominada *node-red-contrib-ui-led*, que simula LEDs que podem ser configurados com diversas cores, indicando o estado lógico de uma variável digital.

A Aplicação de Monitoramento basicamente faz requisições ao microserviço CLP em ciclos predeterminados para obter os dados das variáveis e demonstrá-las graficamente. A Figura 53 ilustra o fluxo construído no Node-RED para o sistema de supervisão das entradas digitais do microserviço CLP.

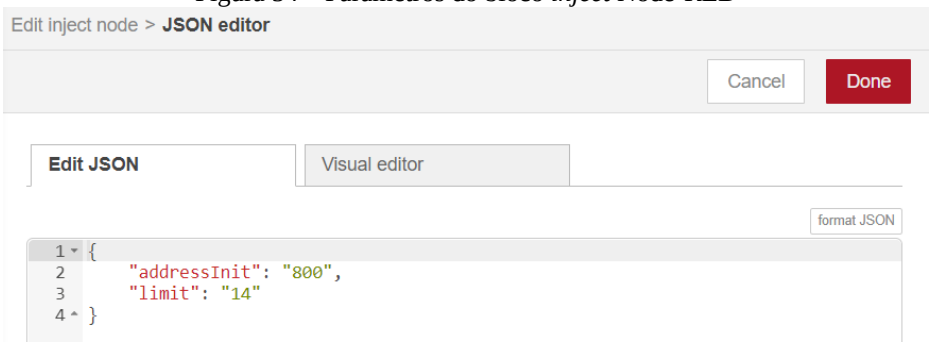
Figura 53 – Fluxo para supervisão das entradas digitais construído no Node-RED



Fonte: Autor

O bloco *inject* dispara as requisições ao microserviço CLP. Ele pode ser configurado para que dispare somente quando acionado, ou em intervalos de tempo, como o caso aqui apresentado. Nesse bloco são inseridos os parâmetros que direcionam ao endereço de monitoramento das entradas digitais do OpenPLC pelo protocolo Modbus TCP. Esses parâmetros são configurados no campo de edição da mensagem (*Payload*) do bloco *inject* em formato JSON, como é apresentado na Figura 54.

Figura 54 – Parâmetros do bloco *inject* Node-RED

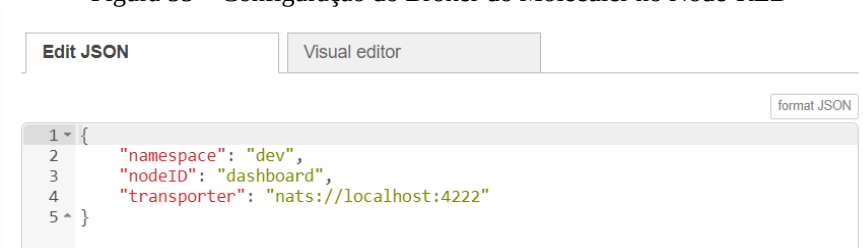


Fonte: Autor

O termo “addressInit” determina o endereço inicial de leitura dos dados das entradas do OpenPLC, e o termo “limit” define a quantidade de endereços a serem lidos a partir do endereço inicial. Isso quer dizer que, a leitura dos dados se inicia no endereço %IX800.0 e termina no endereço %IX800.13, percorrendo, portanto, 14 endereços. É importante observar que os endereços utilizados para a leitura dos dados transmitidos pelo protocolo Modbus TCP no OpenPLC iniciam a partir de 800.0.

O bloco *call* do Moleculer é responsável por fazer chamadas internas aos microserviços. Portanto, através desse método acessamos a rota da ação desejada, que é a ação de leitura das entradas digitais do OpenPLC, de acordo com a Figura 53. Nesse bloco também são configurados os parâmetros do broker para a construção da aplicação, como mostra a Figura 55.

Figura 55 – Configuração do Broker do Moleculer no Node-RED



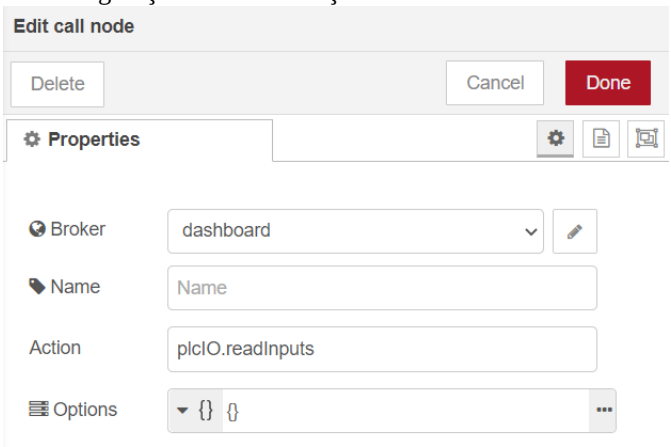
Fonte: Autor

Na configuração do broker “namespace” é o espaço de nome para segmentar os nós na mesma rede (por exemplo: "desenvolvimento", "preparo", "produção"). O termo “*nodeID*” pode possuir qualquer identificação. É importante que essa identificação dos nós nunca seja igual à de outro microserviço, pois isso causaria conflito na operação do sistema.

O termo "*Transporter*" determina qual o tipo de sistema de envio de mensagens será utilizado. Figura 55 é indicado que o *Transporter* é o NATS e a porta de comunicação padrão é a 4222.

Para realizar a leitura das entradas digitais do microserviço CLP é necessário acessar a ação readInputs, portanto, no campo *Action* do bloco *call*, deve ser inserido o nome do microserviço seguido da ação desejada, conforme indica a Figura 56.

Figura 56 – Configuração de acesso à ações do Moleculer através do Node-RED



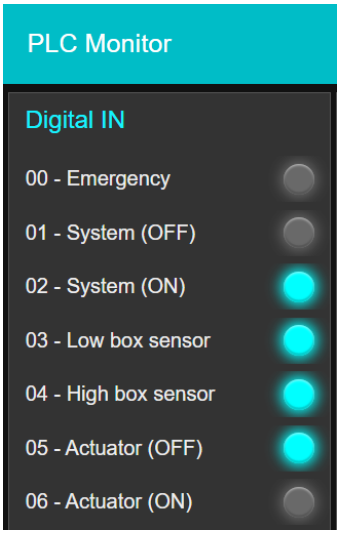
Fonte: Autor

A resposta enviada pela ação readInputs retorna um *array*, contendo os valores das entradas de 0 a 13 do microserviço CLP. O bloco split tem a função de dividir os dados contidos no *array* disponibilizado pelo microserviço CLP. Isso é necessário nessa aplicação para demonstrar o estado lógico das entradas do OpenPLC individualmente em um *dashboard*, através da representação gráfica de LEDs.

O bloco switch está conectado após o bloco split. O switch tem a função de distribuir os dados recebidos do bloco split de forma separada, um dado em cada saída. Portanto os 14 dados vindos do *array* disponibilizado pelo microserviço CLP, dividido pelo bloco split, serão distribuídos 1 em cada saída do switch, sendo assim possível exibir o estado logico de cada entrada do OpenPLC separadamente.

Os blocos led são utilizados para representar graficamente o estado das variáveis de entrada do microserviço CLP no *dashboard*. Esses LEDs gráficos podem ser configurados com diversas cores, que são definidas seguindo o padrão de formatação CSS, sendo uma cor para cada nível lógico. A Figura 57 apresenta o *dashboard* criado para a leitura das entradas do microserviço CLP.

Figura 57 – *Dashboard* desenvolvido no Node-RED



Fonte: Autor