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

JEFERSON ANDRÉ BIGHETI

Arquitetura de Automação e Controle Orientada a Microserviços para a Indústria 4.0

Bauru
2020

JEFERSON ANDRÉ BIGHETI

ARQUITETURA DE AUTOMAÇÃO E CONTROLE ORIENTADA A MICROSERVIÇOS PARA A INDÚSTRIA 4.0

Tese 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 Doutor em Engenharia Elétrica.

Área de Concentração: Automação

Linha de Pesquisa: Mecatrônica

Orientador: Prof. Dr. Eduardo Paciência Godoy

Bauru 2020

B592a

Bigheti, Jeferson André

Arquitetura de automação e controle orientada a microserviços para a indústria 4.0 / Jeferson André Bigheti. -- Bauru, 2020

143 f. : il., tabs.

Tese (doutorado) - Universidade Estadual Paulista (Unesp), Faculdade de Engenharia, Bauru

Orientador: Eduardo Paciência Godoy

1. Arquitetura Orientada Microserviços (MOA). 2. Framework Moleculer. 3. Internet das Coisas Industrial (IIoT). 4. Sistema de Controle via Rede. 5. Comunicação M2M. 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.

UNIVERSIDADE ESTADUAL PAULISTA

Câmpus de Bauru

ATA DA DEFESA PÚBLICA DA TESE DE DOUTORADO DE JEFERSON ANDRÉ BIGHETI, DISCENTE DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA, DA FACULDADE DE ENGENHARIA - CÂMPUS DE BAURU.

Aos 05 dias do mês de março do ano de 2020, às 13:30 horas, no(a) ICT - Sorocaba, reuniu-se a Comissão Examinadora da Defesa Pública, composta pelos seguintes membros: Prof. Dr. EDUARDO PACIÊNCIA GODOY - Orientador(a) do(a) Departamento de Engenharia de Controle e Automação / Instituto de Ciência e Tecnologia - UNESP - Câmpus de Sorocaba, Prof. Dr. PAULO JOSÉ AMARAL SERNI do(a) Departamento de Engenharia de Controle e Automação / Instituto de Ciência e Tecnologia - UNESP - Câmpus de Sorocaba, Prof. Dr. MAURÍCIO BECERRA VARGAS do(a) Departamento de Engenharia de Controle e Automação / Instituto de Ciência e Tecnologia - UNESP - Câmpus de Sorocaba, Prof. Dr. DENIS BORG do(a) Departamento de Engenharia Mecatrônica / Faculdade de Engenharia de Sorocaba (FACENS), Prof. Dr. RUBENS ANDRE TABILE do(a) Departamento de Engenharia de Biossistemas / Universidade de São Paulo / USP, sob a presidência do primeiro, a fim de proceder a arguição pública da TESE DE DOUTORADO de JEFERSON ANDRÉ BIGHETI, intitulada **ARQUITETURA DE AUTOMAÇÃO E CONTROLE ORIENTADA A MICROSERVIÇOS PARA A INDÚSTRIA 4.0**. Após a exposição, o discente foi arguido oralmente pelos membros da Comissão Examinadora, tendo recebido o conceito final: APROVADO.

Nada mais havendo, foi lavrada a presente ata, que após lida e aprovada, foi assinada pelos membros da Comissão Examinadora.

Prof. Dr. EDUARDO PACIÊNCIA GODOY

Prof. Dr. PAULO JOSÉ AMARAL SERNI

Prof. Dr. MAURÍCIO BECERRA VARGAS

Prof. Dr. DENIS BORG

Prof. Dr. RUBENS ANDRE TABILE

Feliz aquele que transfere o que sabe e aprende o que ensina.

(Cora Coralina)

À minha adorada esposa Gláucia, companheira e grande incentivadora, presença indispensável em minha vida e à minha filha, Helena e ao meu filho Matheus (luzes da minha vida).

Agradecimentos

À minha família, em especial à minha esposa Gláucia, que sempre me apoiou e estimulou em todos os momentos.

Gostaria de registrar aqui meus profundos agradecimentos ao meu orientador, Prof. Dr. Eduardo Paciência Godoy pela inspiração, amizade e constante disponibilidade que proporcionou durante todo o programa de Doutorado. É um professor que foi além das obrigações profissionais do magistério, revelando-se uma pessoa iluminada pelo idealismo da excelência na profissão que abraçou. Seus profundos e comprovados conhecimentos técnicos são coroados por sua elevada qualidade moral.

Aos meus pais Osneide Bigheti e Terezinha de Lourdes Bigheti que sempre me proporcionaram todos os recursos necessários para que eu trilhasse meu caminho acadêmico com segurança.

Ao SENAI pela colaboração e apoio.

Aos Diretores do SENAI Prof. Laerte Padilha Lozigia e Prof. Everson de Aro Capobianco pelo incentivo e apoio.

Ao Prof. Sergio Luiz Risso e demais Professores do SENAI pelas discussões e idéias.

Ao Michel de Mattos Fernades e colegas de pós-graduação que tanto me ajudaram nesse percurso.

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

RESUMO

A Indústria 4.0 (I4.0) têm focado no uso conjunto de tecnologias de automação e informática industrial visando a obtenção de maior eficiência, qualidade e produtividade. Para o desenvolvimento da I4.0 é necessário a aplicação de tecnologias recentes como a Internet das Coisas Industrial (IIoT), os Sistemas de Controle via Redes e 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, esta tese foca no desenvolvimento de uma arquitetura orientada a microserviços (MOA) para aplicações de automação e controle com foco na I4.0. A arquitetura é baseada no uso do *framework Moleculer* para microserviços e no desenvolvimento de serviços e composições de serviços para suportar as aplicações requeridas. Os serviços desenvolvidos podem ser executados em diferentes plataformas como computadores, sistemas embarcados e em nuvem. Serviços de Infraestrutura como Comunicação Máquina a Máquina, Controlador Lógico Programável e Aquisição de Dados são responsáveis por prover as funcionalidades básicas da arquitetura e usados para a composição das aplicações. Os serviços de Processos/Negócios como Controle e Supervisão de Processos executam as funcionalidades de alto nível e necessitam operar em composição com os serviços de infraestrutura. A arquitetura MOA, serviços e aplicações foram implementados e validados através de experimentos com malhas de controle em planta didática e em ambiente *Hardware-In-the-Loop* (HIL). Resultados experimentais demonstraram a viabilidade da arquitetura para aplicações de automação e controle de processos. A grande vantagem da arquitetura é a interoperabilidade, fornecida pela comunicação automática e transparente entre os serviços e aplicações, que é um fator importante para a I4.0. Uma análise temporal e da comunicação em rede mostrou que, apesar da latência e do jitter serem significativos, a arquitetura é confiável e recomendada para aplicações de automação e controle com orquestração de serviços com tempo de ciclo de ao menos 300 ms. A arquitetura também forneceu maior modularidade e escalabilidade para aplicações industriais, a partir da replicação e composição dos serviços.

Palavras-chave: Serviços; Framework Moleculer; Internet das Coisas; Sistema de Controle via Rede, Manufatura Avançada; Comunicação M2M.

ABSTRACT

Industry 4.0 (I4.0) has focused on the joint use of automation technologies and industrial informatics aiming at achieving greater efficiency, quality and productivity. The application of some recent technologies such as the Industrial Internet of Things (IIoT), Networked Control Systems and Cloud Computing, among others, is required for the development of the I4.0. The great challenge of I4.0 is to promote the integration of these technologies, equipment and systems allocated in different hierarchical levels of industrial systems. A recent paradigm has been the collaborative automation through the use and sharing of services in order to obtain a flexible, distributed and network-integrated architecture. In this context, this thesis focuses on the development of a Microservice Oriented Architecture (MOA) for automation and control applications focused on I4.0. The architecture is based on the use of the Moleculer framework for microservices and on the development of services and service compositions in order to support the required applications. The proposed services can be executed in different platforms such as computers, embedded systems and in the cloud. Infrastructure services such as Machine-to-Machine Communication, Programmable Logic Controller and Data Acquisition are responsible for providing the basic functionality for the architecture and for composition of applications. Business services such as Control and Supervision of Processes provide high-level functionality and need to operate in composition with the infrastructure services. The MOA architecture, services and applications were implemented and validated through experiments with control loops in a didactic plant and in a Hardware-In-the-Loop (HIL) environment. Experimental results demonstrated the feasibility of architecture for automation and process control applications. The great advantage of the architecture is interoperability, provided by the automatic and transparent communication between services and applications, which is an important factor for I4.0. A temporal analysis and network communication evaluation showed that although latency and jitter are significant, the architecture is reliable and recommended for automation and control applications with cycle time of service orchestration of at least 300 ms. The architecture also provided greater modularity and scalability for industrial applications, from the replication and composition of the services.

Keywords: Services; Moleculer Framework; Internet of Things; Networked Control Systems, Smart Manufacturing, Machine to Machine Communication.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SIGLA	DESCRIÇÃO
ADC	Analog to Digital Converter – Conversor Analógico Digital
AESOP	Architecture for Service-Oriented Process
AMQP	Advanced Message Queuing Protocol
AutomationML	Automation Markup Language
API	Application Programming Interface - Interface de Programação de Aplicações
CC	Corrente Continua
CoAP	Constrained Application Protocol
CLP	Controlador Lógico Programável
CPS	Cyber-Physical Systems – Sistemas Ciber-físicos
CPPS	Cyber-Physical Production Systems – Sistemas de Produção Ciber-físicos
DAQ	Data Acquisition
DDS	Data Distribution Service
DPWS	Device Profile for Web Services
FBD	Function Block Diagram - Diagrama de Blocos Funcionais
HTTP	HyperText Transfer Protocol
I2C	Inter-Integrated Circuit
IoT	Internet of Things – Internet das Coisas
IIoT	Industrial Internet of Things – Internet das Coisas Industrial
I4.0	Industry 4.0 – Indústria 4.0
IL	Instruction List - Lista de Instruções
ISA	International Society of Automation
IP	Internet Protocol
IPv6	Internet Protocol version 6
JSON	JavaScript Object Notation
KPI	Key Performance Indicator – Indicadores Chaves de Performace
LD	Ladder Diagram
MOA	Microservice Oriented Architecture - Arquitetura Orientada a Microserviços
MQTT	Message Queuing Telemetry Transport
M2M	Machine to Machine – Máquina a Máquina
MV	Manipulated Variable – Variável Manipulada
NCS	Networked Control System – Sistema de Controle via Rede
OPC UA	Open Platform Communications Unified Architecture
PaaS	Platform as a Service – Plataforma como Serviço
PC	Personal Computer – Computador Pessoal
PCI	Peripheral Component Interconnect
PID	Proporcional Integral Derivativa
PV	Process Variable – Variável do Processo
PWM	Pulse Width Modulation –Modulação com Largura de Pulso
RFID	Radio Frequency Identification – Identificação por Rádio Frequência
RAAS	Robot as a Service
HIL	Hardware-In-the-Loop
REST	Representational State Transfer
RCP	Rapid Control Prototyping - Prototipagem Rápida de Controle
RPC	Remote Procedure Call

SCADA	Supervisory Control and Data Acquisition
SDCD	Sistema Digital de Controle Distribuído
SIL	Software-In-the-Loop
SFC	Sequential Function Chart - Sequenciamento Gráfico de Funções
SOAP	Simple Object Access Protocol
SOA	Service Oriented Architecture – Arquitetura Orientada a Serviço
STL	Structured Text - Texto Estruturado
TI	Tecnologia da Informação
TA	Tecnologia da Automação
TCP	Transmission Control Protocol - Protocolo de Controle de Transmissão
TIC	Tecnologia da Informação e Comunicação
UDP	User Datagram Protocol - Datagrama Protocolo de Usuário
USB	Universal Serial Bus - Porta Universal
Wi-Fi	Wireless Fidelity
WNCS	Wireless Networked Control Systems – Sistemas de Controle via Redes Sem Fio
WSDL	Web Service Description Language
WS-BPEL	WS – Business Process Execution Language
WS-CDL	WS –Choreography Description Language
XML	eXtensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

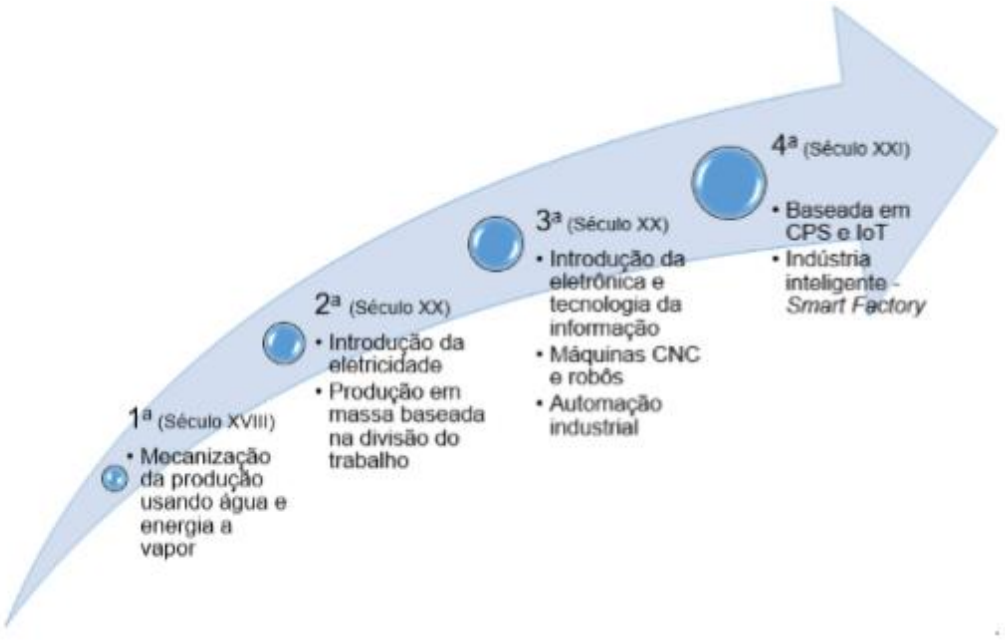
1.1 CONTEXTUALIZAÇÃO

Devido a sua alta capacidade em acelerar processos de produção e manufatura, as tecnologias de automação (TA) tornaram-se essenciais para as indústrias que planejam melhorar seus processos e obter 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 propiciaram a criação de novos métodos de produção nas indústrias globais 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, coordenadas de modo a conferir competitividade ao negócio, otimizar a eficiência da cadeia produtiva, adicionar valor ao produto, racionalizar o uso dos recursos e customizar as soluções tecnológicas é chamada de 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) (BERGER, 2014; (LU, 2014) 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 cedem seu lugar para inteligência descentralizada, num ambiente onde todos os equipamentos e máquinas estão conectadas em redes e disponibilizando informações de forma única (COLOMBO et al., 2014).

A maioria das tecnologias necessárias à implementação da I4.0 já existem atualmente e se encontram em diferentes estágios de maturação. O caráter disruptivo que a I4.0 traz é fruto da articulação e convergência dessas tecnologias. Entre elas podem-se citar (ZUEHLKE, 2010; RAJKUMAR et al., 2010; STANKOVIC, 2014, LEE et al., 2015; HEGAZY & HEFEEDA, 2015):

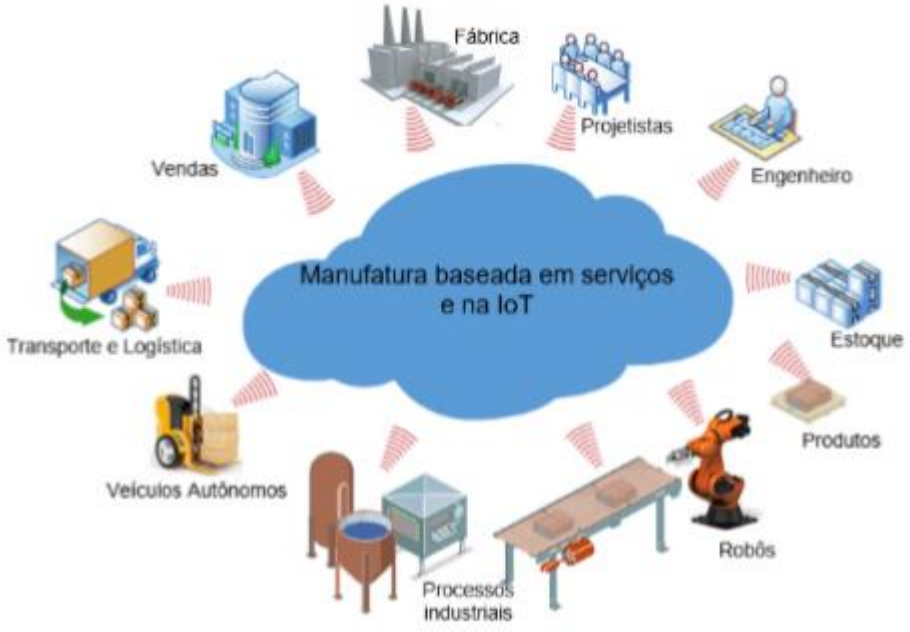
- ✓ Protocolo IPV6 para ampliação dos pontos de conexão IP a todos os equipamentos;
- ✓ Ampliação das redes sem fio para flexibilização da comunicação e aquisição de dados;
- ✓ Sistemas de controle via redes (NCS – *Networked Control Systems*) e sistemas ciber-físicos (*CPS – Cyber Physical Systems*);
- ✓ 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 I4.0, portanto, é promover a integração entre essas tecnologias, visando a obtenção de uma nova realidade produtiva, onde tudo estará conectado para que as melhores decisões de produção, custo e segurança sejam tomadas, tudo sob demanda e em tempo real. Nesse sentido, muitos trabalhos têm focado no desenvolvimento de 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. Na Figura 2 é mostrado a comunicação em nuvem entre "coisas", isto é, entre os elementos envolvidos no sistema produtivo sobre uma infraestrutura baseada em Internet Industrial.

Hegazy et al. (2015) apresentam um novo serviço na nuvem, "automação industrial como serviço", onde os controladores são hospedados em dois servidores em nuvem, fisicamente separados. Nikolaidis et al. (2015) propõem a interação entre dois conceitos: a utilização de controladores em nuvem com dispositivos baseados em IIoT. Chen et al. 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 Arquiteturas Orientada a Serviço (SOA) promissoras, que podem ser utilizadas em Processo de

Monitoramento e Controle com uso do protocolo como *Constrained Application Protocol* (CoAP), *Open Platform Communications Unified Architecture* (OPC UA) e entre outros, para acesso a sensores e atuadores.

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



Fonte: Pisching (2017).

Interoperabilidade e segurança são os dois maiores desafios em torno da implantação da IIoT e da I4.0. Para SISINNI et al. (2018), o maior desafio está em torno da interoperabilidade entre dispositivos e máquinas que usam protocolos e arquiteturas diferentes nos diferentes níveis hierárquicos dos processos industriais. As indústrias estão sendo pressionadas a modernizar os seus sistemas e equipamentos para acompanhar a crescente velocidade e volatilidade do mercado e lidar com tecnologias que se modificam constantemente.

Baseando-se nas necessidades e desafios citados, um paradigma recente para a indústria tem sido o desenvolvimento de soluções colaborativas por meio do uso e compartilhamento de serviços para obtenção de uma arquitetura flexível, escalável, 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). Diante desse contexto, esta pesquisa foca no estudo e desenvolvimento de uma arquitetura baseada em microserviços (MOA - *Microservices Oriented Architecture*), que representa uma evolução da SOA, para aplicações de IIoT e I4.0. Uma arquitetura de microserviços consiste em uma coleção de pequenos serviços autônomos, onde cada serviço é independente e implementa uma única funcionalidade.

1.2 RELEVÂNCIA E JUSTIFICATIVA

As soluções baseadas em Ethernet Industrial permitem ao usuário a integração dos equipamentos industriais a serviços de TI fornecidos por servidores (DECOTIGNIE, 2005). Dessa forma, tornou-se possível disponibilizar informações provenientes dos sistemas de automação por meio de protocolos com suporte ao Ethernet TCP/IP em aplicações Web, essa interconexão de dados, infraestruturas de rede de TI e de automação de forma segura e confiável representa uma tendência para o desenvolvimento de soluções inovadoras e integradoras na área industrial com a IIoT e a I4.0 (BERGER, 2014).

A comunicação sem fio atualmente é de fundamental importância para que a implementação do conceito de Indústria 4.0 se tornem viáveis. Uma das tecnologias base para este conceito é a Internet das Coisas (IoT) e o Máquina para Máquina (*M2M – Machine to Machine*) (STANKOVIC, 2014). A Computação em Nuvem se refere, essencialmente, à noção de utilizar, em qualquer lugar e independente de plataforma, variadas aplicações por meio da Internet com a mesma facilidade de tê-las instaladas em computadores locais (DIVYA & JEYALATHA, 2012). Kushida & Pingali (2014) apresentam uma revisão sobre os requisitos para a aplicação da computação em nuvem na indústria, descrevendo potenciais vantagens dessa tecnologia e discutindo casos de estudo com resultados reais desse tipo de implementação.

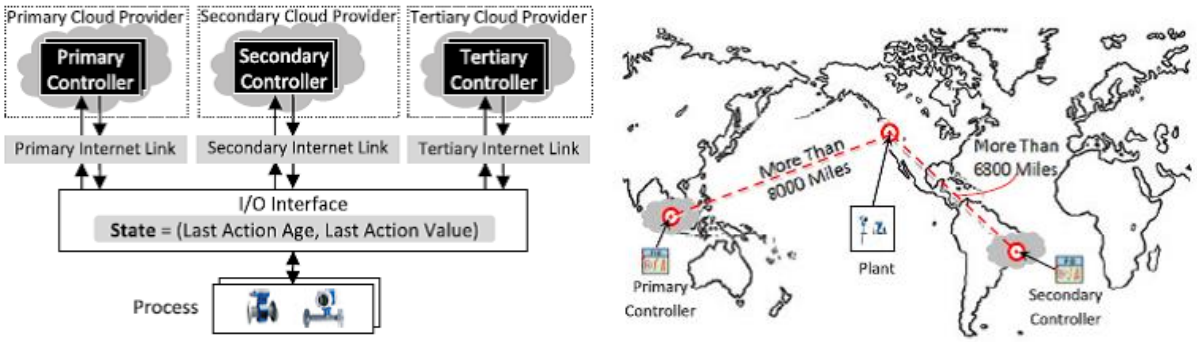
A Internet das Coisas Industrial (IIoT) tem revolucionado a fabricação de produtos, oferecendo oportunidades para as empresas se atualizarem, oferecerem novos serviços, melhorarem suas produções e, consequentemente, permitindo também a aquisição e o acesso de uma quantidade de dados sem precedentes, em velocidades muito maiores e com muito mais eficiência do que antes (SISINNI et al., 2018). A IIoT pode melhorar significativamente a conectividade, a eficiência, a escalabilidade, a economia de tempo e de custos para as organizações industriais. As redes de dispositivos inteligentes da IIoT permitem que as indústrias obtenham dados de toda sua corporação, de forma integrada, obtendo uma visão completa sobre a gestão da empresa e auxiliando na tomada das melhores decisões.

Processos industriais, bem como muitas outras infraestruturas críticas dependem de sistemas *Supervisory Control and Data Acquisition* (SCADA) e Sistema Digital de Controle Distribuído (SDCD) para executar suas funcionalidades complexas. A IIoT é complementar a esses sistemas, utilizando as informações provenientes dos sistemas SCADA e SDCD como fontes de dados. Sistemas SCADA tem foco em monitoramento e controle, enquanto a IIoT busca analisar os dados para aumentar a produtividade e eficiência (KULKARNI, 2016).

Ainda no contexto da IIoT e da I4.0, nota-se a crescente integração entre os sistemas industriais e a nuvem, por meio de alternativas como os Sistemas Ciber-Físicos (CPS - *Cyber*

Physical Systems). Por meio de serviços remotos, busca-se obter maior flexibilidade da produção e ferramentas de monitoramento, otimização de recursos e gerenciamento do processo produtivo. Recentemente, tem-se discutido o potencial da Computação em Nuvem para virtualização de sistemas da I4.0 e aplicações de automação e controle em nuvem. Hegazy & Hefeeda (2015) apresentam uma proposta de arquitetura de um sistema de automação em nuvem, apresentado na Figura 3.

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



Fonte: Hegazy & Hefeeda (2015)

Nesta proposta da Figura 3, diversas aplicações como controle em malha fechada e virtualização de controladores são realizadas na nuvem. Os autores afirmam que a proposta traria inúmeros benefícios em termos de redução de custo e tempo de desenvolvimento por meio da virtualização de controladores, redução de *hardware* utilizado e tempo de desenvolvimento de aplicações. 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.

Colombo et al. (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 arquiteturas orientadas a serviço (SOA) são consideradas um caminho promissor para concepção do modelo de fábrica do futuro. No entanto, existe a necessidade e potencial para melhorar as funcionalidades e minimizar problemas de integração por meio de abordagens colaborativas de sistemas e serviços (COLOMBO et al., 2014).

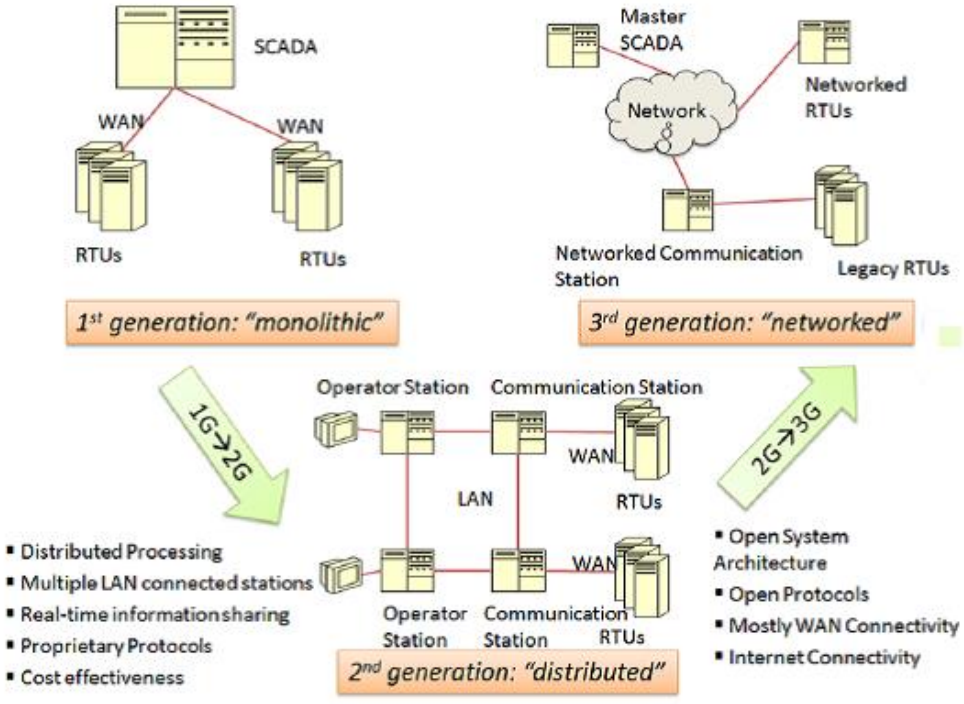
Nesse sentido é importante considerar os próximos passos para a evolução dos sistemas industriais por meio da IIoT e I4.0, de forma que consigam atender os desafios recentes como grau de descentralização e independência dos sistemas. Sendo assim, a proposta de arquiteturas SOA possui grande potencial, a partir da disponibilização de serviços em nuvem alocados em

diferentes dispositivos, *gateways* ou sistemas, os quais facilitam e padronizam as interações entre eles. Leitão et al. (2016) apresentam um estudo sobre novas arquiteturas SOA para aplicações industriais.

A evolução das arquiteturas de automação e controle industriais, mostrada na Figura 4, evidencia a tendência para a convergência do uso das tecnologias de automação e da informação. A primeira arquitetura era centralizada e com comunicação ponto a ponto entre equipamentos. Já a segunda e a terceira incorporaram o conceito de redes de comunicação, permitido a distribuição do controle, redução de custos e facilidade de comissionamento e manutenção.

Adicionalmente, a terceira arquitetura incorporou as chamadas arquiteturas de controle via redes (NCS – *Networked Control Systems*) (Gupta & Chow, 2010), provendo a descentralização do controle, modularização e flexibilização. O uso das redes sem fio em NCS, promovendo interoperabilidade entre redes com fio já existentes e novas redes sem fio nortearam o surgimento dos sistemas de controle via redes sem fio (*WNCS - Wireless Networked Control Systems*) (FISCHIONE et al. 2011).

Figura 4 - Evolução das Arquiteturas de Automação e Controle Industrial

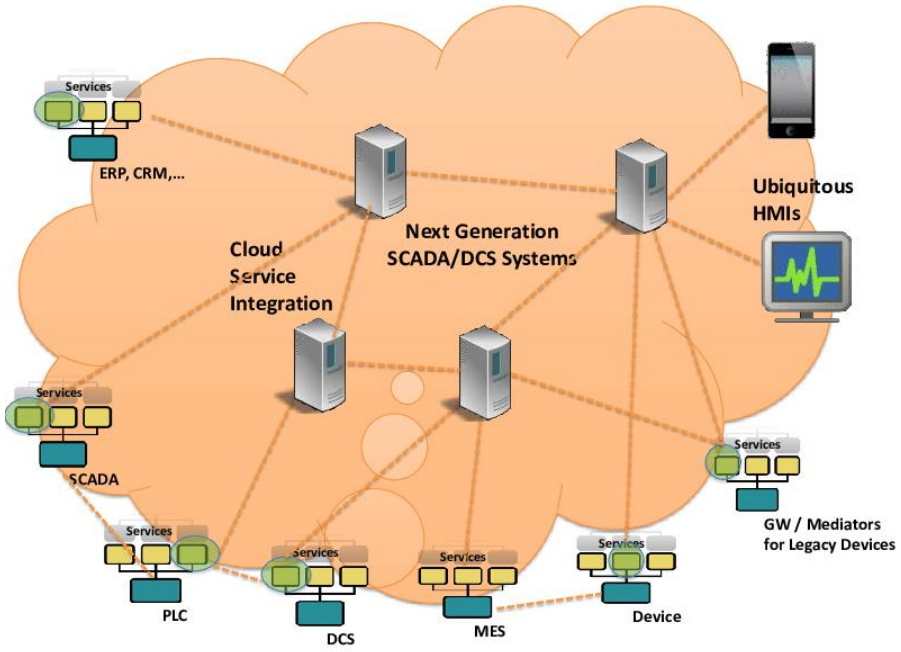


Fonte: Colombo et al. (2014)

A quarta e atual arquitetura, ilustrada na Figura 5, é a baseada nos CPS alavancados pela total integração demandada pela IIoT e I4.0. Nessa nova proposta de arquitetura, um sistema de armazenamento de informações em nuvem será compartilhado pelos equipamentos e

sistemas, tornando possível o uso de serviços padronizados para comunicação entre os mesmos. Os serviços poderão ser acessados por aplicações, sistemas e outros serviços independentemente de onde eles estão alocados (*hardware*, *software* e rede de comunicação), propiciando uma arquitetura colaborativa.

Figura 5 - Nova Arquitetura Baseada em Serviços em Nuvem

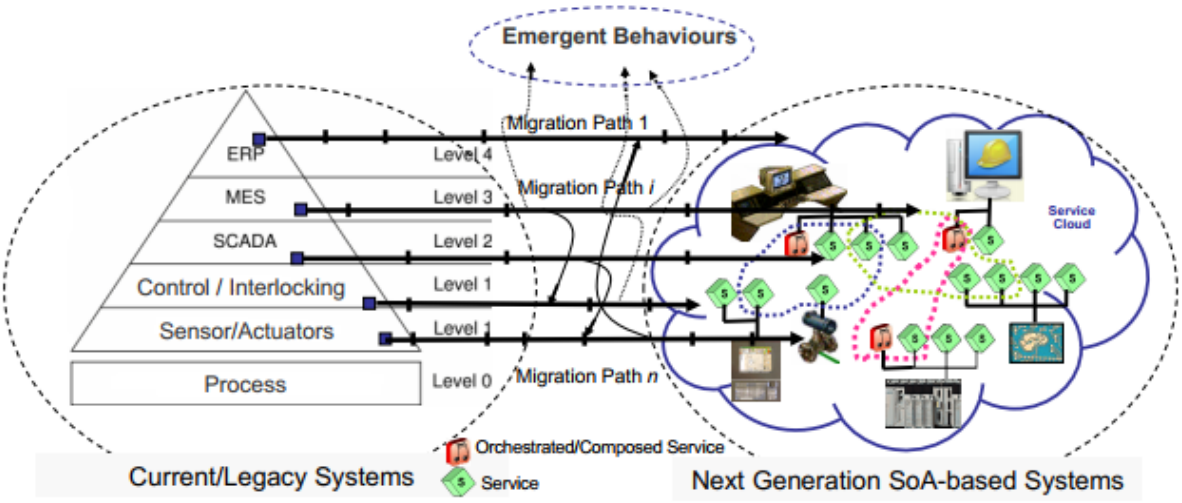


Fonte: Colombo et al. (2014)

Para Leitão et al. (2016), a SOA em aplicações de IIoT e I4.0 é adequada para tratar os seus recursos, principalmente pela natureza heterogênea desses ambientes. 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 operadas por diferentes módulos, de sistemas independentes da heterogeneidade do *hardware*/*software*, permitindo a interoperabilidade dos sistemas como mostrado da Figura 6. Nessa figura verifica-se que novos serviços poderão ser criados a partir dos serviços existentes usando as técnicas de orquestração ou coreografia e há também uma migração da arquitetura 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.

Portanto, a consolidação das tecnologias de automação e informática industrial compatíveis com a ideia de uma arquitetura baseada em serviços em nuvem para aplicações IIoT e I4.0 norteiam essa pesquisa.

Figura 6 - SOA Aplicada na Automação de um Processo



Fonte: Colombo et al. (2017).

1.3 OBJETIVO

Esta pesquisa propõe o desenvolvimento de uma arquitetura de automação e controle orientada a microserviços para aplicações no contexto de Internet das Coisas Industrial e Indústria 4.0. A integração de tecnologias de automação e informática industrial e o uso de um *framework* para microserviços permitirá a criação de aplicações de IIoT e I4.0 no formato de serviços em nuvem, proporcionando maior versatilidade e interoperabilidade com diferentes sistemas industriais.

1.4 RESUMO DA PROPOSTA DO TRABALHO

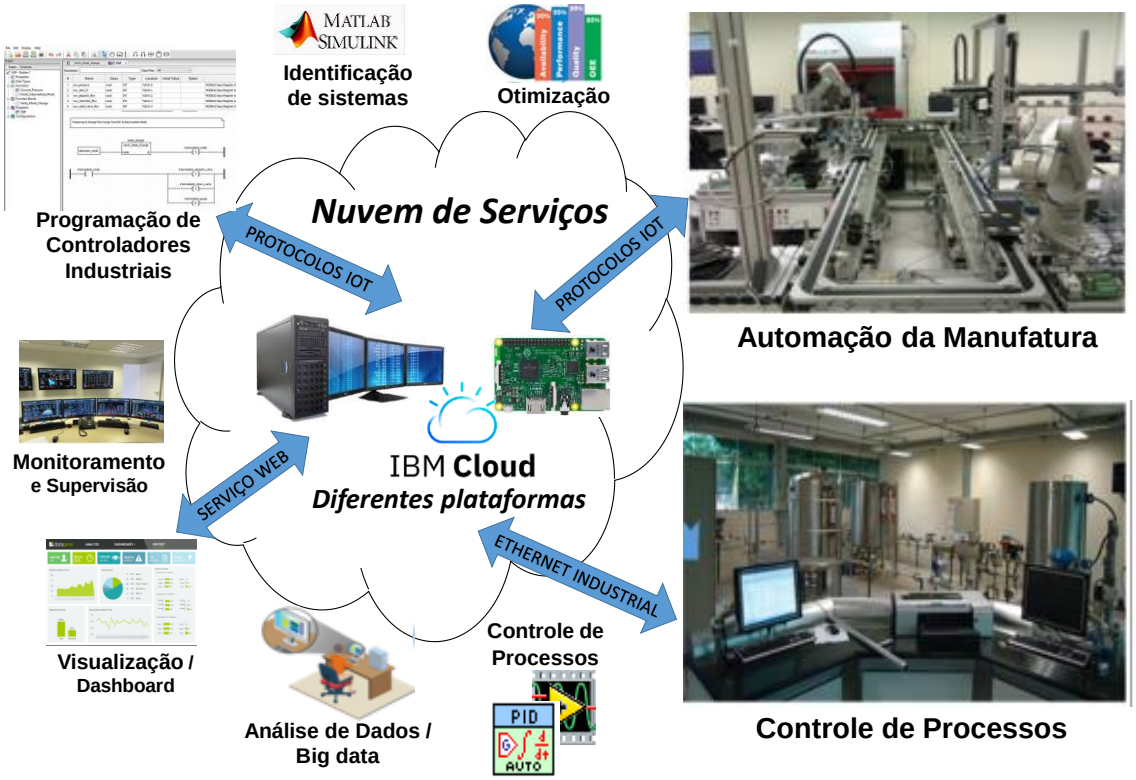
A proposta resumida da arquitetura orientada a microserviços (MOA) para aplicações no contexto de IIoT e I4.0 deste trabalho é mostrada na Figura 7. Diversas tecnologias de automação e informática industrial foram integradas na arquitetura proposta. Subsistemas de automação e controle, como plantas didáticas e ambientes simulados, serão usados para testes e aplicações da arquitetura.

Conforme Wollschlaeger et al. (2017), por meio da utilização integrada de tecnologias de automação, como redes Ethernet Industrial Modbus TCP/IP e o padrão OPC UA, e de TI, como serviços Web no padrão REST, e protocolos de IoT como MQTT e CoAP, torna-se possível a

disponibilização de serviços em nuvem, os quais podem ser utilizados na composição de serviços de alto nível e aplicações relacionadas às tarefas requeridas no contexto de IIoT e I4.0, como monitoramento e supervisão de sistemas, controle e otimização de processos e etc.

De forma resumida, a MOA habilita aplicações da IIoT e I4.0 ao disponibilizar em nuvem, de forma padronizada e interoperável, as informações dos dispositivos da planta/processo (sensores, atuadores, controladores), para qualquer tipo de aplicação (serviço) e plataforma (*smartphones, tablets* ou computadores), tornando os dados da planta/processo e a realização dos serviços disponíveis acessíveis em tempo real, para qualquer ponto conectado à rede. A apresentação detalhada da MOA proposta neste trabalho é realizada na seção 5.1.

Figura 7 – Proposta Geral da Arquitetura Orientada a Microserviços para Aplicações de IIoT e I4.0



Fonte: Autor

1.5 CONTRIBUIÇÕES

Neste trabalho é apresentada uma solução para o problema do uso de serviços virtuais aplicado a processos industriais que faz uso da integração das tecnologias de automação e informática industrial utilizando arquitetura MOA que permitiu o compartilhamento de microserviços para obtenção de uma arquitetura flexível, escalável, interoperável, distribuída e conectada em rede. A arquitetura desenvolvida neste trabalho representa uma mudança de paradigma nas interações entre os diferentes sistemas industriais, equipamentos, aplicações e

usuários. Apesar da estrutura hierárquica tradicional da arquitetura ISA95 ainda ser presente na indústria, a MOA desenvolvida representa uma arquitetura alternativa e complementar de informações que disponibiliza uma grande variedade de serviços, permitindo que as informações provenientes de sistemas heterogêneos possam ser obtidas de forma transparente ao usuário.

1.6 ESTRUTURA DO TRABALHO

Além desse Capítulo 1 introdutório, esta tese de Doutorado apresenta outros nove capítulos. No Capítulo 2 são apresentados os conceitos básicos utilizados no desenvolvimento desta pesquisa como a convergência da tecnologia de automação com a de informática industrial e também a evolução da arquitetura orientada a serviços (SOA). Também é apresentado neste capítulo a arquitetura baseada em microserviços (MOA) como uma alternativa ao tradicional padrão arquitetural monolítico.

No Capítulo 3 são revisadas e discutidas as principais arquiteturas SOA para aplicações da IIoT e I4.0 para encontrar os pontos de convergências para a elaboração da proposta deste trabalho. O *framework Moleculer*, que suporta o desenvolvimento da MOA, é apresentado no Capítulo 4, com descrição dos seus principais componentes e configurações.

Em seguida, no Capítulo 5 é detalhada a proposta da arquitetura orientada a microserviços (MOA) para aplicações de IIoT e I4.0. Neste capítulo é descrita a estrutura geral da MOA que utiliza para o seu desenvolvimento o *framework Moleculer* e discute o desenvolvimento de diferentes microserviços para a realização de atividades base ou de suporte para aplicações de IIoT e I4.0. Os tipos de composições de microserviços da MOA também são apresentados.

O Capítulo 6 apresenta o desenvolvimento da arquitetura, focando na implementação dos microserviços e na composição das aplicações desenvolvidas para este trabalho. Os resultados experimentais que suportam a validação da arquitetura, bem como demonstram sua modularidade e interoperabilidade são apresentados e discutidos no Capítulo 7. Os diferentes experimentos realizados com a operação da MOA em planta didática e em ambiente de simulação em tempo real são detalhados.

O Capítulo 8 apresenta as conclusões parciais deste trabalho e o Capítulo 9 elenca as bibliografias utilizadas como base para o desenvolvimento deste trabalho.

O Capítulo 10 fornece um material complementar, na forma de apêndices, para permitir um maior conhecimento sobre conteúdos complementares ao trabalho.

2 CONCEITOS E TECNOLOGIAS RELACIONADAS

Este capítulo apresenta os conceitos básicos utilizados no desenvolvimento desta pesquisa. Fundamentalmente, as técnicas utilizadas para convergência das tecnologias de automação e informática industrial são discutidas e, também, a evolução da arquitetura orientada a serviços (SOA). A revisão bibliográfica compreendeu pesquisas nas áreas como M2M, Internet das Coisas, Indústria 4.0 e principalmente as arquiteturas orientadas a serviços.

2.1 M2M

Para Kim et al. (2014), o grande desafio da Indústria 4.0 (I4.0) é integrar todos os padrões de comunicação dos ambientes industriais dentro de uma nova proposta onde o protocolo IP passa a ser comum nos ambientes de produção e desta forma permita a integração dos padrões de comunicação. Esse movimento de integração teve início com a proposta do conceito de comunicação M2M. Segundo Stankovic (2014), M2M representa a interconexão entre dispositivos ou sistemas, onde tais sistemas podem trocar informações entre si, de forma autônoma, e tomar decisões de produção e segurança relacionadas ao processo por meio de um modelo de inteligência gerenciado por um dispositivo microcontrolado. M2M engloba dispositivos específicos com a capacidade de coletar dados em um determinado ambiente. Os dados gerados a partir da comunicação entre máquinas podem ser armazenados e utilizados por uma aplicação. As comunicações M2M nasceram de tecnologias desenvolvidas para possibilitar a comunicação totalmente autônoma entre dispositivos e equipamentos sem qualquer intervenção humana. As aplicações pioneiras foram a telemetria e o sistema de monitoramento industrial SCADA. Na Tabela 1 são apresentados alguns exemplos de comunicação M2M.

Tabela 1 - Aplicações para M2M.

Aplicação	Descrição
Segurança e Proteção	Controle de acesso físico, monitoramento de ambiente, sistemas de vigilância
Utilidades	Eletricidade, água, gás, medição
Cuidados com a saúde	Monitoramento de sinais vitais, diagnóstico remoto, telemedicina
Manutenção e Controle	Automação industrial, sensores, luz, bomba
Veicular	Informação de tráfego, diagnóstico remoto, gestão de frotas

Fonte: Adaptado de Kim et al. (2014).

Cackovic et al. (2014) descrevem em seu trabalho que os dispositivos M2M formam uma rede local, que pode ser desde uma rede residencial até uma grande rede industrial. Os dados gerados trafegam da rede local para a rede de comunicação através do *gateway*, que é

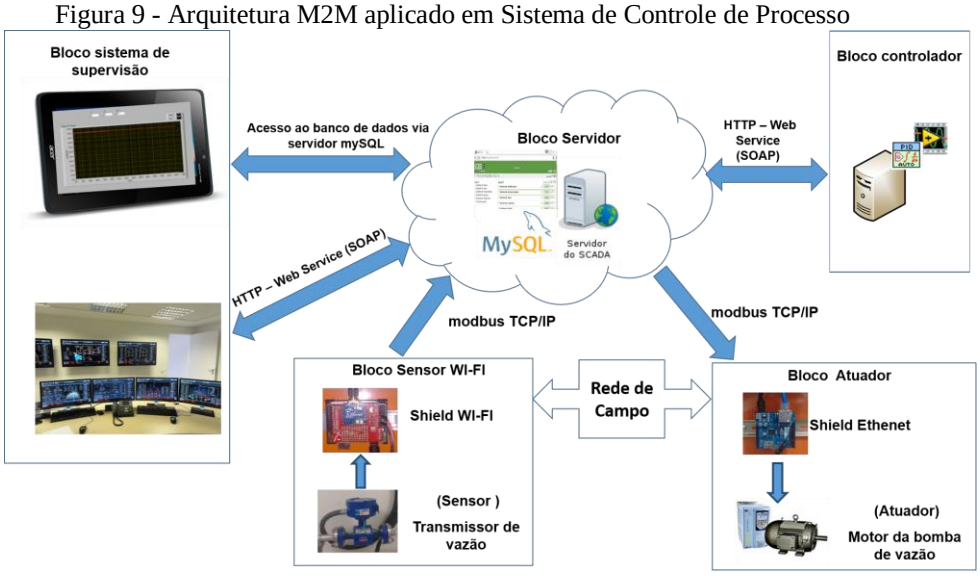
responsável pela conversão dos dados da rede local em algum formato adequado para a rede de comunicação. Uma camada *middleware*, que encaminha os dados e converte os formatos de dados, também pode fazer parte do núcleo da rede, onde se localiza o *gateway* como pode ser visto na Figura 8. A camada *middleware* também permite funções de gerenciamento como autenticação e notificações.

Figura 8 - Arquitetura M2M

Fonte: Adaptado de Cackovic et al. (2014).

Em Bigheti et al. (2016) é apresentado uma plataforma M2M aplicado a uma arquitetura de automação e controle de processos em nuvem, mostrado na Figura 9. O sistema de controle proposto se comunica via Modbus TCP/IP e o servidor Web é implementado usando a ferramenta *open source* ScadaBR. Este estudo de caso apresenta o controle via rede de uma malha de vazão por meio de um controlador com comunicação via *Web Service* com o servidor de dados.

Figura 9 - Arquitetura M2M aplicado em Sistema de Controle de Processo



Fonte: Bigheti et al., 2016.

2.2 INTERNET DAS COISAS (IOT)

A IoT é um paradigma que preconiza um mundo de objetos físicos embarcados com sensores e atuadores, conectados por redes de comunicação, principalmente sem fio, e que se comunicam usando a Internet, moldando uma rede de objetos inteligentes capazes de realizar variados processamentos, capturar variáveis e reagir a estímulos externos (FERREIRA, 2018). Na IoT, há vários tipos de dispositivos de comunicação, também são chamados de “coisas”, que trocam informações entre equipamentos, produtos e objetos pessoais (GUBBI et al., 2013; KASTNER et al., 2014). A infraestrutura de rede de comunicação da IoT pode ser dinâmica e autoconfigurável e é baseada em protocolos de comunicação interoperáveis onde objetos físicos e virtuais, possuem identificadores, atributos físicos e identidade virtual, fazendo uso de interfaces inteligentes para integração (KRAMP, 2008).

O termo IoT foi inventado por Kevin Asthon em 1999 no contexto de *Supply Chain* da indústria. Porém, a definição se expandiu para outras aplicações, como na área de saúde, transporte, comunicação (SUNDMAEKER et al., 2010). Atualmente, as pesquisas em torno da IoT têm atraído interesse e focado em diferentes aplicações (STANKOVIC, 2014).

Cada foco de aplicação da IoT possui características e requisitos específicos. O *Consumer IoT* representa as aplicações da IoT centrados no homem. Nessas aplicações, os dispositivos inteligentes de consumo são interconectados entre si a fim de melhorar a consciência humana sobre o ambiente ao seu redor. Em geral, as comunicações de IoT referentes ao consumo podem ser classificadas como de máquina para usuário e na forma de interações cliente-servidor. Quando aplicada à área industrial, a *Industrial IoT* busca simplificar e criar arquiteturas de sistemas que são mais acessíveis, responsivas e efetivas, com comunicações eficientes e interação entre a produção e sensores, atuadores, entre outros, para aprimorar o desempenho e flexibilidade da indústria (LYDON, 2014).

A Tabela 2 faz uma comparação qualitativa dessas tecnologias.

Tabela 2 - Comparação entre IoT e IIoT.

	Consumer IoT	**Industrial IoT**
Impacto	Revolução	Evolução
Modelo de serviço	Voltada para o ser humano	Orientada as maquinas
Estado Atual	Novos dispositivos e padrões	Dispositivos e padrões existentes
Conectividade	Estruturas de rede não estruturada do tipo Ad- Hoc ou móvel	Estruturado (nós fixos; centralizado com gerenciamento de rede)
Criticidade	Não rigorosa	Muito rigorosa em função do tempo, confiabilidade, segurança e privacidade.
Volume de dados	De médio para alto	De alto para muito alto

Fonte: Adaptado de SISINNI et al., (2018).

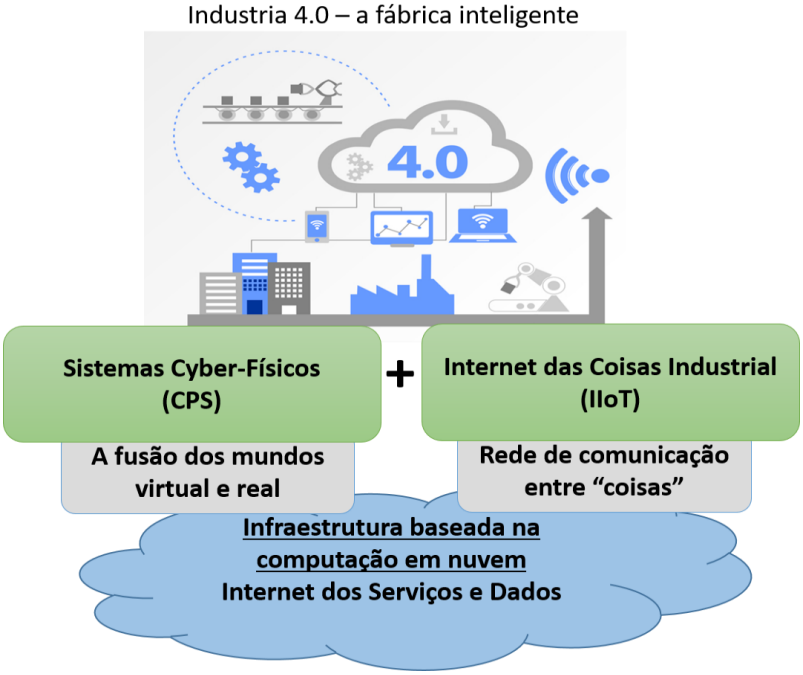
Em relação à conectividade e criticidade, a *Consumer IoT* é mais flexível, permitindo estruturas de rede do tipo Ad-Hoc ou móvel, e tendo requisitos de tempo e confiabilidade menos rigorosos. Por outro lado, a *Industrial IoT* normalmente emprega soluções de rede fixas e baseadas em infraestrutura que são bem projetadas para as necessidades da comunicação industrial. Na IIoT, as comunicações são na forma de máquinas para máquinas (M2M) onde os requisitos são rigorosos em termos de confiabilidade (SISINNI et al., 2018).

Os dispositivos de IoT são sistemas heterogêneos que objetivam conectar diferentes "coisas" por meio das redes de comunicação, exigindo assim tecnologias para os integrar e assim torná-los compatíveis. Xu; He & Li (2014) e Souit et al. (2013) afirmam que a tecnologia para integrar esses dispositivos da IoT é a SOA.

2.2.1 Internet das Coisas Industrial (IIoT)

Para Xu et al. (2014), a introdução da Internet das Coisas a Serviços no ambiente de produção está inaugurando uma revolução industrial. A aplicação da IoT nos segmentos industriais e de fabricação é conhecida como IIoT e é uma tecnologia fundamental para a implantação da I4.0 (HE et al., 2016). Este novo tipo de indústria é baseado no modelo de *Smart Factory* (WANG et al., 2016), como pode ser visto na Figura 10.

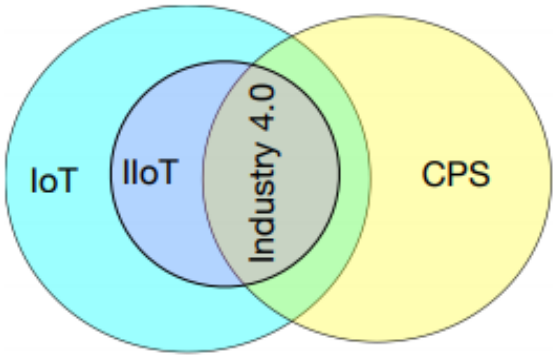
Figura 10 - Os CPSs, a IIoT, os Serviços e a Computação em Nuvem no Contexto da I4.0



Fonte: Adaptado de Pisching et al. (2015a)

As fábricas estão se desenvolvendo em ambientes inteligentes nos quais a distância entre o mundo real e o digital está se tornando menor. Em resumo, a IIoT é um subconjunto da IoT, especificamente para aplicações industriais. No ciclo de vida de fabricação de um produto é onde ocorre a interseção entre o sistema CPS com a IoT e com o seu subconjunto IIoT originando a Industria 4.0. A Figura 11 apresenta interseções de IoT, CPS, IIoT e Industria 4.0 (SISINNI et al., 2018).

Figura 11 - Interseções de IoT, CPS, IIoT e I4.0



Fonte: Sisinni et al. (2018).

Os novos conceitos da IIoT e I4.0 permitirão a transição para fábricas inteligentes com redes de comunicação interligando todos equipamentos da automação. Para Weyer et al. (2015), os principais aspectos IIoT aplicados na Indústria 4.0 podem ser especificados através de três paradigmas:

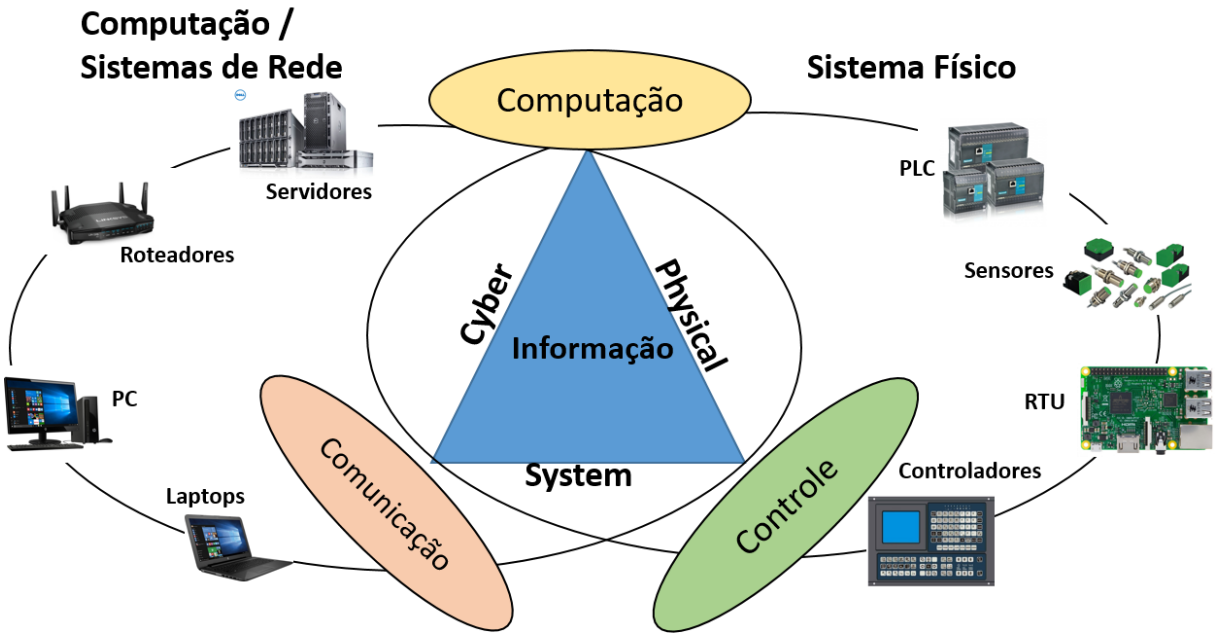
- ✓ O Produto Inteligente;
- ✓ A Máquina Inteligente;
- ✓ Operador Aumentado.

A ideia do Produto Inteligente (*Smart Product*) é estender o papel do produto a uma parte ativa do sistema. As características dos produtos são armazenadas diretamente em banco de dados com os requisitos operacionais do plano de construção individual do produto. O paradigma da Máquina Inteligente (*Smart Machine*) descreve o processo de máquinas que se tornam Sistemas de Produção Ciber-Físicos (*CPPS*). O terceiro paradigma mencionado acima, o Operador Aumentado (*Augmented Operator*), visa dar suporte tecnológico ao trabalhador que é a parte mais flexível no ambiente dos sistemas de produção altamente modulares com uso tablets, óculos inteligentes e relógios inteligentes (Weyer et al., 2015).

2.3 SISTEMAS CIBER-FÍSICOS (CPS)

O termo “Sistema Ciber-Físico (CPS) l (CPS)” foi proposto em 2006 por um grupo de pesquisadores da fundação americana NSF (*National Science Foundation*), com o propósito de descrever a integração entre dispositivos de controle com a computação e redes de comunicação como pode ser visto na Figura 12. Os CPS são sistemas com capacidade computacional que estão conectados com dispositivos do mundo físico e os processos que os cercam, ao mesmo tempo que fornecem e consomem serviços de dados disponíveis na Internet (KANG et al., 2016). De acordo com Colombo et al. (2017), os dispositivos de campo, máquinas, módulos de produção e produtos são compreendidos como CPS que trocam informações de forma autônoma, acionando ações e controlando umas às outras de forma independente. Para Hermann et al. (2016), o CPS, para se adequar aos requisitos da I4.0, deve abranger clientes, máquinas, produtos, estoques e prestadores de serviço, de forma que interajam executando ações autonomamente.

Figura 12 - Integração entre Dispositivos de Controle com a Computação e Comunicação num CPS



Fonte: Autor.

2.4 INDÚSTRIA 4.0 (I4.0)

O desenvolvimento das tecnologias digitais propiciou a criação de novos métodos de produção nas indústrias globais baseados na automação, robótica, sistemas ciber-físicos, inteligência artificial, internet das coisas e inteligência de dados, dentre outras inovações. A

utilização dessas tecnologias no contexto industrial, coordenadas de modo a conferir competitividade ao negócio, otimizar a eficiência da cadeia produtiva, adicionar valor ao produto, racionalizar o uso dos recursos e customizar as soluções tecnológicas é chamada de Indústria 4.0 (KAGERMANN, WAHLSTER & HELBIG, 2013).

O termo Indústria 4.0 foi apresentado pela primeira vez em 2011 na Feira de Hannover e após isso, em outubro de 2012, um grupo de trabalho apresentou um conjunto de recomendações de implementações ao governo alemão. Esse termo iniciou a partir de um projeto para estratégia de alta tecnologia do governo alemão, o qual defendeu a informatização da indústria. A partir desse conceito, as empresas apresentaram soluções apoiadas pelos governos, principalmente o Europeu (especialmente Alemão), mas também por países como os Estados Unidos, Brasil, Japão e China, indicando um ponto estratégico nesta era industrial. I4.0 no Brasil é conhecido como Manufatura Avançada, e os estudos das tecnologias envolvidas começaram a acontecer por volta de 2013. Nos Estados Unidos, o conceito recebeu o nome de *Smart Industry* (Indústria Inteligente) (A VOZ DA INDUSTRIA, 2016).

De acordo com Colombo et al. (2017), a I4.0, dispositivos de campo, máquinas, módulos de produção e produtos são compreendidos como Sistemas Ciber-Físicos (CPS) que trocam informações de forma autônoma, acionando ações e controlando umas às outras de forma independente e também interagindo entre si para controlar sistemas e processos de forma distribuída e com grande capacidade de tomada decisões, seja de forma autônoma ou de forma colaborativa. Kagermann, Wahlster & Helbig (2013) descrevem sobre o uso de estratégias para a implementação da I4.0, visando manter a Alemanha como uma das principais produtoras de máquinas e tecnologia, além de possibilitar que suas indústrias continuem figurando entre as mais avançadas do mundo.

Segundo Kagermann Wahlster & 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.

Conforme Wang, Towara & Anderl (2017), a inovação técnica da I4.0 baseia-se na integração vertical e horizontal de sistemas industriais, engenharia digital contínua ao longo do ciclo de vida do produto e a descentralização de recursos computacionais. Para que isso ocorra, é necessário um modelo comum que agrupe tais condições dentro de uma arquitetura de referência para a I4.0 (ADOLPHS et al., 2015). Em consequência dessas necessidades, em abril de 2015 foi apresentado na feira de Hannover, um Modelo de Referência de Arquitetura para a Indústria 4.0, denominado RAMI 4.0 (*Reference Architectural Model for Industry* 4.0).

O RAMI 4.0 compatibiliza os elementos inseridos na I4.0 em um modelo tridimensional, organizado em camadas, onde a partir dessa estrutura, tecnologias baseadas na I4.0 podem ser classificadas e desenvolvidas (HANKEL, 2015). No entanto, apesar de sua grande aceitação, alguns estudos relatam algumas limitações em termos de praticidade com relação ao RAMI 4.0. Wang, Towara & Anderl (2017) salientam que o modelo em si é um pouco abstrato, generalizando de forma demasiada a sua aplicação.

2.5 AUTOMAÇÃO E INFORMÁTICA INDUSTRIAL

A IIoT é uma tecnologia fundamental para a implantação da I4.0. As novas aplicações de IIoT e I4.0 têm demandado nos últimos anos uma maior aplicação de tecnologias de TI como sistemas Web, Computação em Nuvem, Serviços Web e arquiteturas orientadas a serviços (*SOA – Service Oriented Architecture*) em sistemas de automação e controle industrial. Também pode ser verificado o surgimento de novas tecnologias como *Devices Profile for Web Services* (DPWS), *Open Platform Communications Unified Architecture* (OPC UA), *Data Distribution Service* (DDS) e o *AutomationML* (*Automation Markup Language*) para suprir as demandas desse novo formato da indústria (JAMMES et al., 2014).

A Ethernet Industrial compreende uma vertente de melhorias e novos desenvolvimentos para cumprir os requisitos de comunicação industrial e viabilizar o uso da Ethernet TCP/IP na indústria (DECOTIGNIE, 2009). O uso do Ethernet Industrial e seus protocolos como Modbus TCP/IP, Profinet e Ethernet/IP viabilizam a interconexão de dados e infraestruturas de redes de TI e TA de forma segura e confiável e tem sido bastante usada para o desenvolvimento de soluções inovadoras e integradoras para IIoT e I4.0 (BERGER, 2014). Algumas dessas soluções envolvem a aplicação dos padrões utilizados por Web Services em camadas cada vez mais baixas da hierarquia industrial, chegando até mesmo ao nível de dispositivos, com padrões como o OPC UA e o DPWS.

OPC UA é um padrão para comunicação industrial do tipo cliente-servidor. É o sucessor do OPC Classic ou OPC DA (*Data Access*), sendo utilizado para a comunicação flexível em aplicações industriais que não tem requisitos críticos de tempo real. O OPC Classic foi criado em 1996, numa iniciativa de usar a tecnologia da Microsoft chamada OLE (*Object Linking and Embedding*) para aplicações em controle de processo (*OPC – OLE for Process Control*). Com a adoção do padrão em outras áreas, além do controle de processos, e a flexibilização para uso em outros sistemas operacionais (Linux, Android, etc), a OPC Foundation mudou sua nomenclatura para *Open Platform Communications* em 2011 (OPC, 2017). O OPC UA é um padrão importante para as aplicações da IIoT e I4.0 (HOPPE, 2017). O servidor fornece acesso a dados e funções com os quais os clientes podem interagir por meio de um conjunto de serviços padronizados.

Colombo et al. (2017) avaliaram os padrões OPC UA e DPWS quanto à sua aplicabilidade em uma SOA no nível de dispositivos. DPWS foi apresentado como uma proposta para uso de protocolos relacionados à tecnologia de *Web Services* na camada de dispositivos de uma rede industrial. A aplicação de *Web Services* a nível de dispositivos tem diversos benefícios, como a unificação dos protocolos utilizados no nível de TI e chão de fábrica, possibilitando a comunicação direta entre o nível de dispositivos e o nível de MES ou ERP da empresa. Além dos serviços desenvolvidos pelo usuário, também são oferecidos serviços relacionados à eventos e infraestrutura, como descoberta de dispositivos utilizando UDP, troca de dados entre dispositivos utilizando mecanismos como o WSDL (*Web Services Description Language*) via *Web Services* SOAP (*Simple Object Access Protocol*). Os autores concluem que nenhuma dessas duas especificações preenchem todos os requisitos de uma SOA a nível de dispositivos, sendo proposta uma combinação dos dois padrões para a obtenção de uma solução mais completa.

A maioria das tecnologias de *Web Services* utilizada na indústria é baseada em SOAP/XML (*eXtensible Markup Language*). No entanto, existe uma vertente visando reduzir

a complexidade dos serviços utilizando o REST (*Representational State Transfer)*. REST é um estilo de arquitetura usada em aplicações descentralizadas orientadas a recursos, que define o uso de um conjunto fixo de interfaces de serviço para representações de recursos heterogêneas.

Ungurean, Gaitan & Gaitan (2016) desenvolvem uma arquitetura para a IIoT utilizando DDS como *middleware*. O objetivo é integrar a maior quantidade possível de protocolos a nível de dispositivos ao sistema. A grande vantagem da utilização de DDS como *middleware* é a possibilidade de configuração dos parâmetros de QoS de modo a alcançar o desempenho de tempo real exigida por uma aplicação industrial, uma vantagem sobre especificações OPC.

CoAP (*Constrained Application Protocol*) é um protocolo de comunicação criado por um grupo de trabalho da *Internet Engineering Task Force* (IETF) chamado *Constrained RESTful Environments* (CoRE). O objetivo do CoRE foi o desenvolvimento de um *framework* para aplicações simples, como monitoramento de sensores, em redes com largura de banda limitada. O CoAP é baseado nos mesmos princípios do modelo REST. Servidores disponibilizam recursos em um URL e os clientes acessam esses recursos usando os mesmos métodos do HTTP (*GET*, *PUT*, *POST* e *DELETE*). Como o HTTP e o CoAP compartilham o modelo REST, eles podem ser facilmente conectados.

MQTT (*Message Queuing Telemetry Transport*) é um protocolo de comunicação que apresenta como principal característica o fato de ser leve e aplicável em redes com alta latência e largura de banda limitada (MARTINS; ZEM, 2015). O MQTT é baseado na arquitetura *publish/subscribe*, em que existem dispositivos que publicam informações na rede, organizando essas informações em categorias ou tópicos, e dispositivos que declaram interesse em determinadas categorias de informação, recebendo assim apenas as informações relevantes para si. Um dispositivo central funcionando como servidor, chamado *broker*, é responsável pela comunicação entre produtores e consumidores de dados. Os dispositivos se comunicam com o *broker* usando TCP (MARTINS & ZEM, 2015).

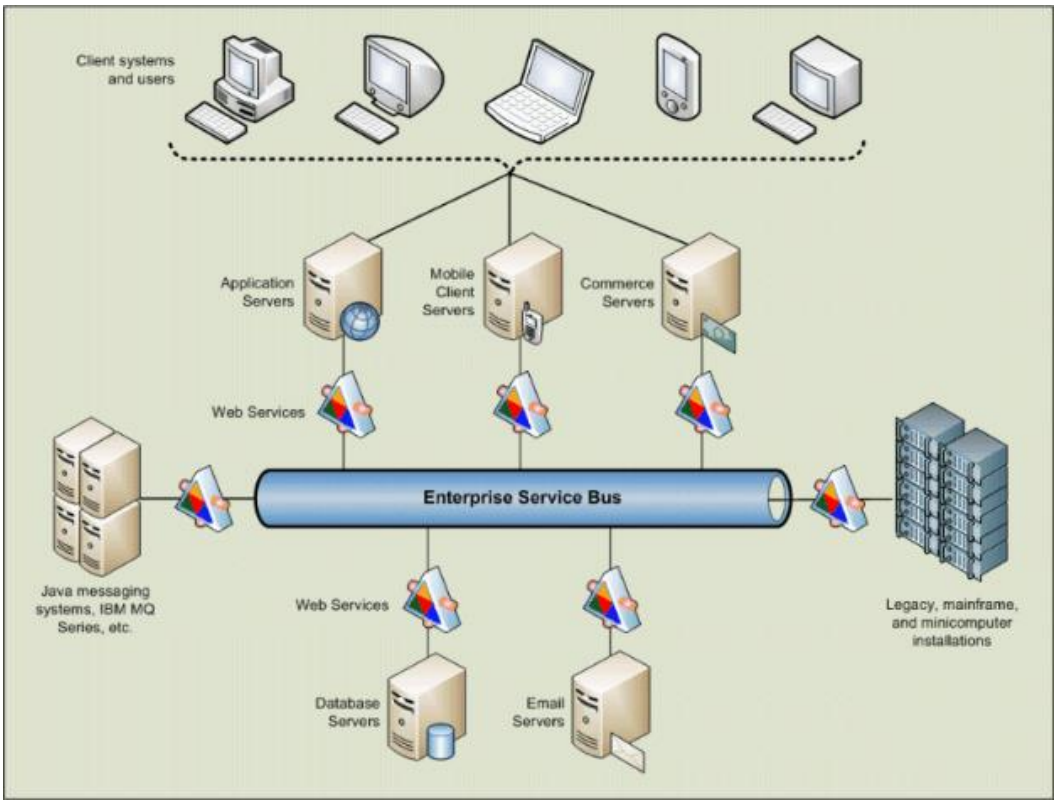
2.6 ARQUITETURA ORIENTADA A SERVIÇOS (SOA)

A arquitetura orientada a serviços (SOA) é uma evolução da computação distribuída. O conceito foi proposto pela primeira vez, no artigo "*Service Oriented Architectures*", em abril de 1996, escrito pelos pesquisadores Roy Schulte e Yefim Natis do Gartner Group. A SOA é um estilo de arquitetura de *software* cujo princípio fundamental prega que as funcionalidades implementadas pelas aplicações devem ser disponibilizadas na forma de serviços.

Para Xiao et al. (2016), estes serviços são conectados por meio de um "barramento de serviços" (*enterprise servisse bus*) que disponibiliza interfaces, ou contratos, acessíveis por

meio de *Web Services* ou outra forma de comunicação entre aplicações, conforme mostrado na Figura 13. Segundo Jammes et al. (2014), a SOA é baseada nos princípios da computação distribuída e utiliza o paradigma requisição/resposta para estabelecer a comunicação entre os sistemas clientes e os sistemas servidores que implementam os serviços.

Figura 13 - SOA Tradicional



Fonte: Hutchison (2016)

SOA não é definida como uma tecnologia, nem como metodologia ou serviço, mas como um conceito de arquitetura para sistemas corporativos, com a finalidade de promover a integração entre o modelo de negócio e a TI por meio de serviços (XIAO et al., 2016), conforme a Figura 13. Esta ponte permite expor as funcionalidades dos aplicativos em serviços padronizados e inter-relacionados. Sendo esse realizado por meio de interfaces de serviços fracamente acoplados, onde os serviços não necessitam de detalhes técnicos da plataforma dos outros serviços para a troca de informações ser realizada.

Muitas pesquisas, como as de Xiao et al. (2016) e Jammes et al. (2014), têm focado no desenvolvimento de SOA com a promessa de oferecer uma arquitetura para suportar a propagação e a utilização de serviços virtuais reutilizáveis. Para Sheng et al. (2014), a SOA é uma arquitetura para organizar infraestruturas e aplicações de *software* em um conjunto de serviços passíveis de interação. Nesse contexto, as informações e os recursos ficam disponíveis

para todos os clientes da organização como serviços independentes que são acessados de um modo padrão.

Já nas pesquisas de Melo (2011), Souit (2011; 2013) e Souit et al. (2013), a SOA estabelece uma arquitetura que permite que os serviços sejam publicados, descobertos e consumidos por aplicações ou outros serviços. Portanto, a ideia básica é assegurar um ambiente uniforme para oferecer, descobrir, interagir e utilizar as aplicações. Nesse caso a SOA permite definir a arquitetura distribuída com o conceito de serviço, onde eles podem ser invocados de forma independente, por qualquer cliente (externos ou internos) que solicitou o serviço, para processar funções simples ou trabalhar em conjunto por meio de implementações coordenadas, com o objetivo de desenvolver novas funcionalidades para os processos existentes.

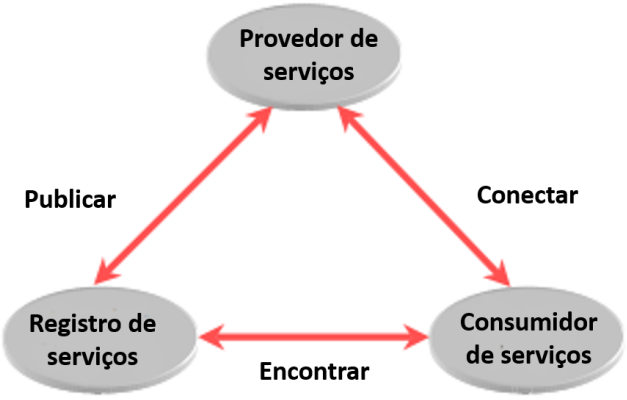
O funcionamento básico de uma SOA consiste na interação de agentes que possuem os papéis de provedor e consumidor de serviços. Neste sentido, os provedores são responsáveis por disponibilizar os serviços que serão utilizados pelos consumidores. Papazoglou (2006) descreve em seu trabalho que a SOA é composta por três elementos básicos, os quais são descritos na Figura 14:

✓ **Provedor de serviços:** responsável por proporcionar a infraestrutura de acesso (via rede) e responder às requisições internas (Intranet) e externas (Internet).

✓ **Consumidor de serviços:** é o cliente do provedor de serviços, podendo ser uma pessoa, organização, máquina ou um componente de software.

✓ **Registro de serviços:** gerencia os repositórios de informações sobre os serviços e organizações que os fornecem, determinando o comportamento que a organização deve ter para divulgar o serviço e como o cliente deve proceder para localizar o serviço desejado.

Figura 14 - Modelo de operação da SOA.



Fonte: Autor

Para desenvolver uma aplicação utilizando a SOA são necessárias as seguintes implementações da arquitetura:

✓ **Contrato de Serviço:** é através do contrato que um serviço expressa o seu propósito e suas capacidades, isto permite que os serviços interajam entre si e sejam requisitados pelos consumidores. Manter o Contrato padronizado é um princípio aplicado dentro do paradigma de design de serviços, que é fornecido por meio de um padrão, para que os serviços sejam registrados em inventários. Um contrato de serviço representa um artefato fundamental, no qual através dele, serviços interagem uns com os outros e com potenciais consumidores. Portanto, existe uma forte necessidade de padronizar os contratos de serviços, a fim de tornar os serviços reutilizáveis, recompostos e interoperáveis.

✓ **Baixo acoplamento**: refere-se ao nível de interdependência entre os serviços, sua implementação e os consumidores do serviço. Na prática, busca-se reduzir o nível de acoplamento para se obter um maior nível de reuso dos serviços. Dentro do paradigma de design, baixo acoplamento é um princípio de projeto que é aplicado aos serviços para garantir que o contrato de serviço não esteja fortemente ligado aos consumidores, nem com a lógica de serviço subjacente e a sua implementação. Isso resulta em contratos de serviços que possam ser evoluídos livremente, sem afetar os consumidores de serviços ou a sua implementação.

✓ **Abstração:** este princípio se baseia na ocultação do maior número de detalhes subjacentes de um serviço e tem muita influência no nível de acoplamento dos serviços. No contrato de serviço que contém detalhes sobre o que é, pode acabar sendo utilizado de uma maneira particular devido ao alto conhecimento sobre o funcionamento do serviço por parte do consumidor. Isso pode afetar a evolução do contrato de serviço, pois o consumidor está indiretamente acoplado à implementação dele, que poderá precisar ser substituído no futuro. De certa forma, isso aumenta o acoplamento do tipo consumidor-contrato, que, embora seja um tipo positivo de acoplamento, demasiadamente impacta de forma negativa na evolução, tanto do prestador de serviços como o consumidor.

✓ **Capacidade de reuso:** é um dos princípios mais enfatizados na orientação a serviços, pois permite redução de tempo e trabalho através do aproveitamento da lógica dos serviços na composição de novas soluções para diversos domínios. Serviços reutilizáveis são projetados para que sua lógica seja independente de qualquer processo de negócio específico ou tecnologia. Reutilização relaciona-se quando o serviço é utilizado para automatizar vários processos de negócio. Essa reutilização elimina a necessidade de criar um novo serviço completo e se torna uma parte de vários processos de negócios, sem fazer parte de qualquer processo de negócio particular. O princípio de reutilização de serviços aborda esses equívocos, fornecendo um conjunto de diretrizes que ajudam a projetar serviços que contêm uma lógica não ligada a qualquer processo de negócio em particular e, portanto, poderiam ser reutilizados em toda a empresa para a automatização de vários processos de negócio. Aplicação com composição de

serviços, captação dos mesmos e aqueles levemente acoplados ajudam a desenvolver serviços combináveis.

✓ **Autonomia do serviço:** refere-se ao nível de independência de um serviço. Sua aplicação visa minimizar a ação das influências externas imprevisíveis, as quais os serviços estão sujeitos. Autonomia de serviços é um princípio que visa fornecer serviços com independência de seu ambiente de execução. Isso resulta em maior confiabilidade, já que os serviços podem operar com menos dependência de recursos sobre os quais há pouco ou nenhum controle. Confiabilidade é crítico para garantir a longevidade do serviço.

✓ **Independência de estado:** por meio deste princípio, busca-se garantir a disponibilidade do potencial dos serviços em escala, pois os serviços devem ser projetados para manterem informações de estados apenas quando realmente forem necessárias. Interação entre *softwares* ou requisitos de negócio muitas vezes exigem o esforço de manter e gerenciar o controle do estado das informações entre as operações. Isso se torna mais importante em arquiteturas distribuídas onde o cliente e o servidor não estão fisicamente na mesma máquina. Numa composição de serviço, um pode armazenar dados específicos da atividade em memória enquanto ele espera um outro serviço completar seu processamento. Gestão eficiente da atividade de serviço relacionado aos dados se torna mais importante como um serviço que visa a reutilização do mesmo. As diretrizes da SOA são construir serviços desacoplados, deslocando a sobrecarga de gerenciamento de estado dos serviços para um outro componente/*middleware* externo. Isso ajuda ainda mais na escalabilidade global da solução.

✓ **Visibilidade:** os contratos dos serviços devem ser publicados e disponibilizados permitindo a descoberta e o acesso pelos seus possíveis consumidores. O potencial de reutilização de um *software* não pode ser conseguido se não souber de sua existência, por isso ser detectável é algo muito importante. Não somente detectável, mas que também seja corretamente compreendido, ou seja, depende da qualidade da meta-informação. No caso de uma solução orientada a serviços, por causa da ênfase dada à reutilização de serviços, é muito claro que as oportunidades devem existir para a sua reutilização, o que só é possível se forem descobertos. Para tornar os serviços detectáveis, algumas atividades são necessárias:

- Documentar o serviço de forma consistente;
- Armazenar a informação documentada em um repositório pesquisável;
- Habilitar a procurar da informação documentada de uma maneira eficiente.

Além de aumentar o potencial de reutilização dos serviços, o mecanismo de descoberta também é necessário para evitar o desenvolvimento de uma solução que já está contida em um serviço existente. Para projetar os serviços que não são apenas detectáveis, mas também fornecer informações interpretáveis sobre suas capacidades, o princípio de descoberta do

serviço fornece diretrizes que poderiam ser aplicadas durante a fase de análise do processo de prestação de serviços.

✓ **Composição:** com base neste princípio, os serviços devem ser projetados de modo que permitam fazer parte de composições de serviços, pois isto promove a criação de novas soluções apenas com a reorganizações dos serviços em novas composições. No desenvolvimento de software, a partir de componentes independentes, incentivam o conceito de composição. Este é o um conceito implícito na orientação aos objetos, onde o produto final é composto de vários objetos interligados, os quais têm as capacidades de se tornarem parte de múltiplas soluções. O mesmo conceito de composição é utilizado em SOA por meio de um processo de negócio automatizado, por meio da combinação de vários serviços,

2.6.1 Serviços Virtuais

Um serviço é uma atividade ou conjunto de atividades de natureza intangível que é o relacionamento entre um provedor e um consumidor. A interação acontece em respostas síncronas ou assíncronas. Na prestação de serviços, existe um fornecedor que proporciona algum tipo de serviço e o usuário que consome o serviço concedido.

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 do serviço seja conhecida. Na TI os serviços são virtuais, sendo esses, processos independentes que são descritos, disponibilizados, localizados e invocados por meio de redes de comunicação. Essas inovações dos serviços por meio de redes de comunicação variam de simples respostas a requisições de processos até a execução de procedimentos de negócios complexos, os quais requerem uma comunicação entre provedores e consumidores de serviços virtuais.

Dustdar & Papazoglou (2006) afirmam que os serviços virtuais estão associados à computação orientada a serviços, onde esses são elementos fundamentais para o desenvolvimento de aplicações e soluções que utilizam as plataformas de computação distribuída. Os serviços virtuais permitem que as organizações exponham suas principais competências programaticamente através da Internet (ou Intranet) usando linguagens e protocolos padrões como XML ou JSON (*JavaScript Object Notation*) por meio de redes de comunicação.

Os Serviços também podem ser construídos considerando a invocação de outros 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

(Erl, 2007). Segundo Dustdar & Papazoglou (2008), as técnicas mais utilizadas para composição de serviços são a orquestração e a coreografia.

2.6.2 Web Services

Web Service consiste na disponibilização de um serviço pela Internet que pode ser acessado em qualquer lugar. Os clientes enviam requisições com informações bem definidas e recebem respostas que podem ser síncronas ou assíncronas. A disponibilização de um serviço se dá por meio de um contrato, que é uma interface que disponibiliza as suas funcionalidades, com uma infraestrutura leve e desacoplada de plataforma que facilita a integração em diferentes tecnologias, como:

- ✓ Protocolo HTTP: transmissão de dados pela Internet;
- ✓ XML: formato padrão para troca de informações, os dados são separados por *tags*;
- ✓ SOAP: fornece uma estrutura padrão de empacotamento para transporte de documentos XML pela internet;
- ✓ WSDL: tecnologia XML que descreve de forma padronizada a interface de um *Web Service*;
- ✓ UDDI (*Universal Description, Discovery and Integration*): descreve um registro de serviços e serve com integração, propaganda e descoberta de serviços.
- ✓ REST: estilo de arquitetura que define um conjunto de serviços e propriedades baseados em HTTP.

Web Services é a tecnologia mais comum para a implementação de SOA por ser uma solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes. É possível com esta tecnologia que novas aplicações possam interagir com aquelas que já existem e que sistemas desenvolvidos em plataformas diferentes sejam compatíveis. Outro ponto importante desta tecnologia, descritos por Fattori et al. (2011b) e Melo (2011), está na solução que agrupa um conjunto básico de padrões para a troca e transporte das mensagens e a coordenação de serviços.

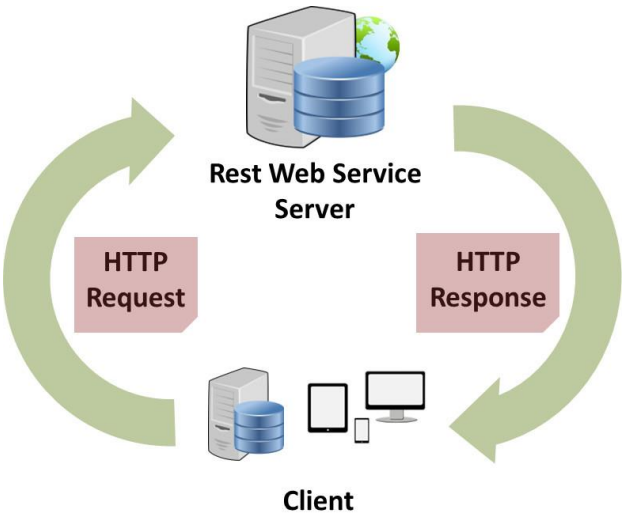
Os *Web Services* são componentes que permitem às aplicações enviar e receber dados em formato de protocolos padrões como XML ou JSON. Neste contexto, *Web Service* é essencialmente definido como um conjunto de atividades computacionais envolvendo inúmeros recursos, cuja intenção é atender necessidades de consumidores ou requisitos de negócios (SHENG et al., 2014; XU et al., 2014).

No caso da I4.0, segundo Colombo et al. (2017), a tecnologia recomendada para a implementação e integração de serviços em SOA também são os *Web Services*, principalmente

pelo fato de permitir a implementação de serviços independente de tecnologia, garantindo a interoperabilidade dos sistemas.

Na Figura 15 é apresentado o ciclo de funcionamento a partir da requisição por um cliente e a resposta da *Web Service*.

Figura 15 - Ciclo de Operação de um Web Service.



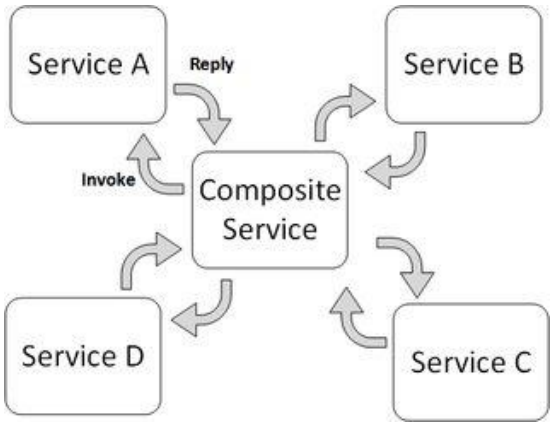
Fonte: Aitoufkir (2016).

2.6.3 Orquestração de Serviços Virtuais

Orquestração de serviços é a composição de serviços para criar um novo serviço, para resolver uma tarefa de um processo de negócio ou para realizar uma atividade de uma aplicação. Neste caso, sempre há a figura de um ponto central, como um maestro de uma orquestra. Orquestração coordena a chamada de outros serviços para compor uma função de maior granularidade. Para Sheng et al. (2014), a orquestração pode ser considerada uma abordagem baseada no modelo *workflow* (Fluxo de Trabalho), contendo nós e o fluxo de dados entre eles, onde um processo central controla os *Web Services* envolvidos e coordena a execução das diferentes operações.

A Figura 16 apresenta um exemplo de orquestração de serviços onde representa um único processo de negócios executável centralizado (o orquestrador) que coordena a interação entre os diferentes serviços. O orquestrador é responsável por invocar e combinar os serviços. A relação entre todos os serviços participantes é descrita por um único ponto de extremidade (ou seja, o serviço composto). A orquestração inclui o gerenciamento de transações entre serviços individuais e ela emprega uma abordagem centralizada para a composição do serviço.

Figura 16 - Exemplo de Composição de Serviços por Orquestração.



Fonte: Risseto (2017).

Apesar de facilitar a implementação, administração e monitoramento, a orquestração de serviços possui limitações de escalabilidade. A orquestração de Serviços Web se dá por meio do padrão WS-BPEL (*Web Services – Business Process Execution Language*) que também inclui o gerenciamento de transações entre serviços individuais, além da manipulação de exceções e erros, assim como a descrição de todo o processo. Em Souit (2013), a orquestração por meio da arquitetura orientada a serviços garante a flexibilidade estrutural considerando as características heterogêneas dos componentes de um sistema produtivo que interagem com sistemas produtivos dispersos.

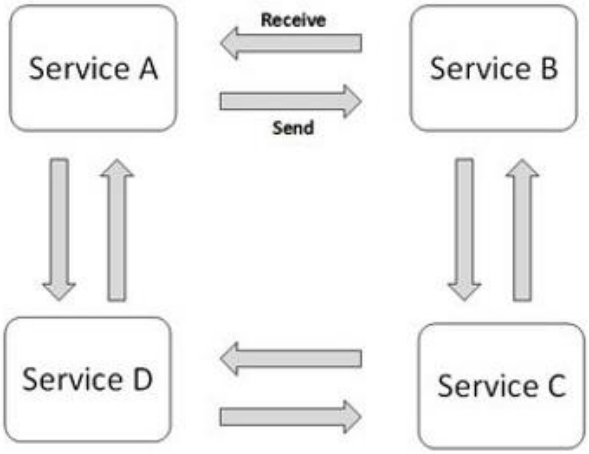
2.6.4 Coreografia de Serviços Virtuais

Na coreografia, a sequência de execução dos serviços já está pré-determinada antes da sua execução e os serviços se conhecem e conversam entre si diretamente sem um intermediador. Por exemplo, quando um serviço é acionado e envia uma mensagem, outros serviços podem estar programados de antemão para receber ou não essa mensagem e dispararem outras ações. Chamamos esse processo de evento.

Essa coreografia de serviço é uma descrição global dos serviços participantes, que é definida pela troca de mensagens, regras de interação e acordos entre dois ou mais terminais. A coreografia emprega uma abordagem descentralizada para a composição do serviço, descrevendo as interações entre múltiplos serviços. Isso significa que uma coreografia difere de uma orquestração, quando relacionado com a lógica que controla as interações entre os serviços envolvidos. Os serviços são acionados conforme a classe de eventos que ocorrem, sendo isso uma característica básica da arquitetura orientada a eventos, onde por meio de um *middleware* é possível atribuir essa característica mediante a criação de fluxos *Publish/Subscribe* como apresentado na Figura 17.

Para Fattori et al. (2011), a coreografia de serviço é mais colaborativa e permite que cada parte envolvida possa descrever seus serviços na interação. A coreografia representa uma descrição global do comportamento de cada um dos serviços participantes da interação, o que é definido pela troca pública de mensagens, regras de interação e acordos entre dois ou mais processos de negócios (SHENG et al.,2014).

Figura 17 – Exemplo de Composição de Serviços por Coreografia.



Fonte: Risseto (2017).

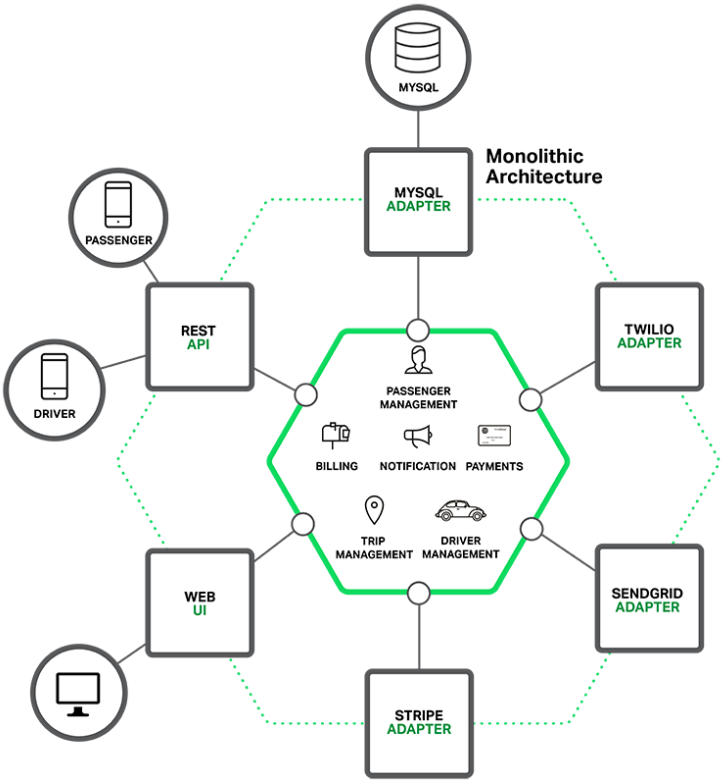
Souit (2013) afirma que a coreografia é a interação entre componentes distribuídos sem a existência de uma entidade controlando a lógica de colaboração, como acontece no mecanismo de orquestração. A coreografia é normalmente associada com interações que ocorrem entre múltiplos Serviços Web, ao passo que apenas as trocas de mensagens públicas são consideradas relevantes e cada serviço conhece apenas sobre suas interações e comportamentos. A coreografia é suportada pelo padrão WS-CDL (*Web Services Choreography Description Language*) (SHENG et al., 2014).

2.6.5 Microserviços

O grande marco dessa arquitetura foi o trabalho escrito por Fowler (2014), onde estão descritas todas as características que definem a estrutura de microserviços. Xiao et al. (2014) descrevem em seu trabalho que o microserviço é um aplicativo flexível, escalável, interoperável, distribuído e totalmente integrado através de redes. Numerosos conceitos têm sido discutidos e novas formas de se organizar e construir sistemas computacionais vêm sendo colocadas em prática, deixando de lado as formas tradicionais de se desenvolver uma aplicação. A arquitetura baseada em microserviços surge como uma alternativa ao tradicional padrão arquitetural monolítico, que é apresentado na Figura 18.

Em uma arquitetura monolítica, uma única aplicação é responsável por todos os processos, sendo que essa aplicação é autossuficiente e independente de outras aplicações. A ideia é que a aplicação seja responsável não apenas por uma determinada tarefa, podendo também executar todos os passos necessários para completar uma macro função. De acordo com a Figura 18, verifica-se que todos os serviços estão alocados no mesmo recurso computacional, indicando certo nível de dependência de um serviço para outro e provendo baixa modularidade para a aplicação. Com o crescimento da aplicação e sua complexidade, havendo um aumento dos serviços acoplados, a dependência entre os serviços começa a dificultar a manutenção e o desenvolvimento de novas funcionalidades (RICHARDSON, 2016).

Figura 18 - Arquitetura Monolítica

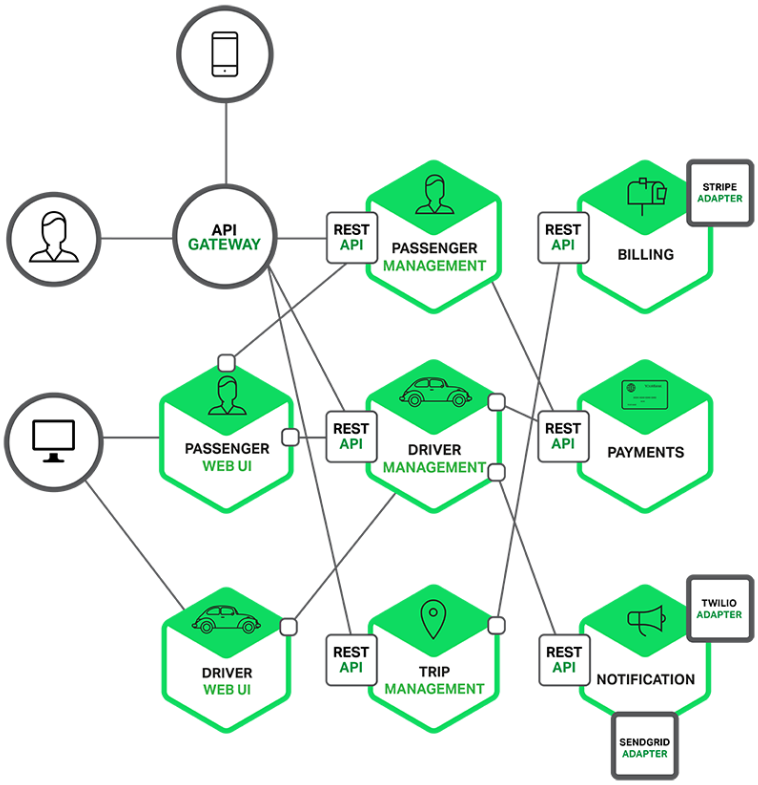


Fonte: Richardson (2016).

A arquitetura baseada em Microserviços (MOA) surge como uma alternativa ao tradicional padrão arquitetural monolítico. A Figura 19 apresenta a estrutura de uma arquitetura orientada a microserviços (MOA). Variante da SOA, os microserviços são um estilo arquitetônico em que as aplicações são decompostas em serviços, oferecendo modularidade, tornando as aplicações mais fáceis de desenvolver, testar, implantar e, o mais importante, alterar e manter. Microserviços também possuem algumas complexidades, como a dificuldade de manutenção de uma base de dados consistente, devido ao fato delas serem distribuídas por cada processo.

Newman (2017) e Richardson (2016) definem os microserviços como pequenos serviços autônomos que trabalham juntos. Esses serviços rodam em seus próprios processos e se comunicam por meio de mecanismos leves tanto síncronos (*request/response*), quanto assíncronos (*request/callback*), geralmente por meio de REST. Os microserviços são implantados e escalados de forma independentes e possuem fronteiras ou limites bem definidos, além de poderem ser escritos em diferentes linguagens e utilizar diferentes recursos para armazenamento de dados.

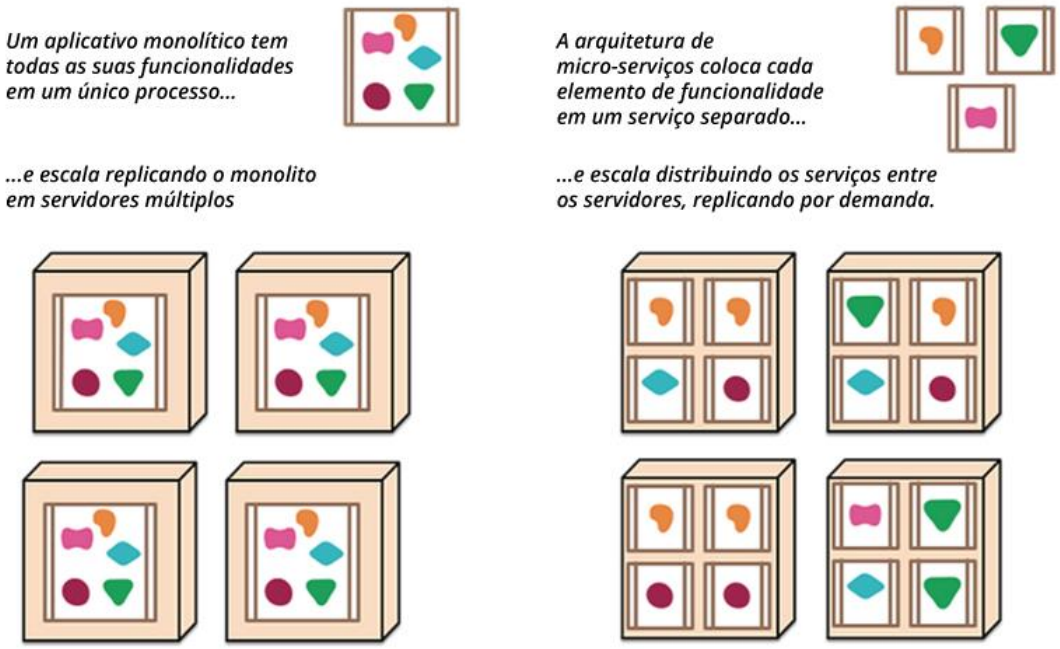
Figura 19 - Arquitetura Orientada a Microserviços



Fonte: Richardson (2016).

Um comparativo entre as arquiteturas monolíticas e microserviços é apresentado na Figura 20. Como pode ser visto, o maior ganho na adoção de microserviços é a escalabilidade. Observa-se na Figura 20, no lado esquerdo, que um ambiente construído em meio à abordagem monolítica dificulta a escalabilidade de serviços específicos, enquanto no lado direito os serviços podem ser clonados de forma a facilitar o aumento de acesso a eles diminuindo a carga de requisições em um mesmo servidor, como também ser capaz de lidar com falhas. Os recursos podem ser melhor aproveitados, evitando desperdício de recursos com alocação de serviços não ou pouco utilizados.

Figura 20 - Comparativo de Estruturas Monolíticas e de Microserviços.



Fonte: Adaptado de Lewis & Fowler (2014)

Características e Vantagens

Fowler (2014) e Newman (20015) descrevem todas as características que definem a estrutura de microserviço. Não é obrigatório encontrar todas essas características em todos os sistemas que seguem esta arquitetura, mas a maioria delas estarão presentes. As principais características que definem um microserviço são:

- ✓ Cada microserviço implementa um único recurso de negócios ou funcionalidade;
- ✓ Um microserviço suficientemente 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 qual 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.

Numa MOA, a saída de um serviço é usada como uma entrada para outro em uma coreografia de serviços independentes. 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. Uma MOA oferece os seguintes benefícios de aplicação:

1. Aumento da resiliência: usando microserviços toda a aplicação é descentralizada e desacoplada em serviços que atuam como entidades separadas;
2. Escalabilidade: A escalabilidade é um aspecto chave dos microserviços, como cada serviço é um componente separado, permite-se expandir uma única funcionalidade (ação) ou serviço sem ter que dimensionar toda uma aplicação;
3. Disponibilidade: os serviços críticos para a aplicação podem ser implantados em vários nós ou servidores para aumentar a disponibilidade e o desempenho de um serviço sem impactar o desempenho de outros serviços;
4. Flexibilidade de desenvolvimento: usando microserviços, o desenvolvimento da aplicação não fica limitado a um único *software* ou linguagem. Dessa forma, é possível escolher a ferramenta certa para cada tarefa;
5. Facilidade de desenvolvimento e modularidade: a criação dos microserviços é mais simples e rápida por executarem funcionalidades específicas, fornecendo maior modularidade para a aplicação.

2.7 SÍNTESE DO CAPITULO

Este capítulo apresentou uma revisão sobre os principais conceitos e tecnologias relacionados à pesquisa como M2M, IoT, IIoT, CPS e principalmente SOA e sua aplicação industrial. Foram apresentados também os conceitos relacionados à serviços e tipos de composição de serviços existentes em arquiteturas SOA. Uma discussão sobre as características e vantagens de uma arquitetura orientada a microserviços foi realizada, buscando esclarecer as diferenças dessa arquitetura para o padrão tradicional monolítico de desenvolvimento de SOA.

3 APLICAÇÕES INDUSTRIAIS DE SOA

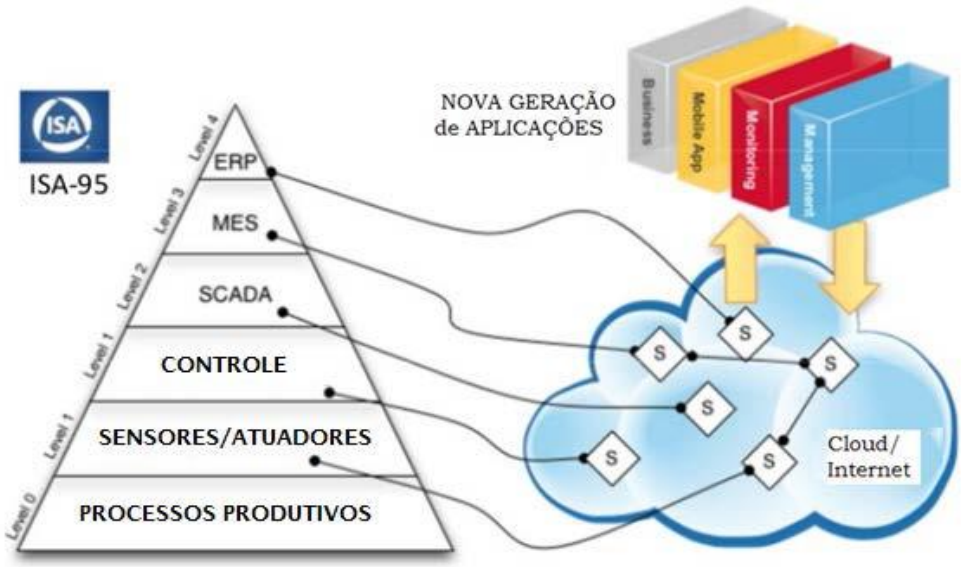
Neste capítulo são apresentadas e discutidas as principais Arquiteturas Orientadas a Serviço (SOA) aplicadas a sistemas industriais para a implementação da IIoT na I4.0, com a finalidade de encontrar os pontos de convergência para a elaboração da proposta deste trabalho.

3.1 TRABALHOS RELACIONADOS

Os avanços tecnológicos relacionados à IIoT e I4.0 estão fundamentos no uso integrado de uma série de tecnologias de automação e informática industrial. Nesse novo modelo de produção industrial, torna-se essencial o conceito de colaboração entre os diversos componentes de uma aplicação e seus níveis hierárquicos no processo industrial. O paradigma da automação colaborativa (GORBACH & MICK, 2002) objetiva o desenvolvimento e implementação de ferramentas e metodologias para fornecer flexibilidade, reconfigurabilidade, escalabilidade e interoperabilidade entre dispositivos e sistemas distribuídos e decentralizados no ambiente industrial. A viabilidade deste paradigma está fundamentada principalmente na adoção de SOA, suportada pelas tecnologias de Computação em Nuvem e serviços virtuais (JAMMES et al., 2014).

Com o uso de SOA, informações provenientes de diversos sistemas heterogêneos podem ser obtidas de forma transparente ao usuário ou aplicação por meio de serviços. Dessa forma, como mostrado na Figura 21, ocorre uma migração da tradicional arquitetura em camadas ISA-95 para uma arquitetura SOA em nuvem.

Figura 21 - Migração da Arquitetura Industrial ISA-95 para SOA



Fonte: Moraes (2017)

Na SOA, cada elemento dos diferentes níveis hierárquicos da arquitetura ISA-95 não mais se comunica somente com as camadas adjacentes e é somente um fornecedor de dados para a camada superior. O conceito de serviços permite que esses elementos passem a ser clientes ou servidores, a depender da necessidade do processo ou aplicação, e que haja interação, baseada em troca de mensagens, entre quaisquer elementos dos níveis hierárquicos do processo industrial (MORAES, 2017). Além disso, a 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.

Diversos trabalhos discutem os benefícios de SOA para aplicações industriais (JAMMES et al., 2014; BLOMSTEDT et al., (2014), (2014); LEITÃO et al., 2016; ESPÍ-BELTRAN, GILART-IGLESIAS & RUIZ-FERNANDEZ, 2017). Para Jammes et al. (2014), SOA é uma arquitetura padronizada que permite acoplamento mínimo para comunicação distribuída entre dispositivos e sistemas, independente de protocolos. SOA fornece mecanismos para a descoberta, abstração, autonomia e composição de serviços que se baseiam em padrões, fornecendo uma contribuição importante para as aplicações industriais (LEITÃO et al., 2016). A facilidade de criação e reuso de serviços fornecida pelo SOA promove modularidade, disponibilidade e escalabilidade para as aplicações industriais (ESPÍ-BELTRAN, GILART-IGLESIAS & RUIZ-FERNANDEZ, 2017).

No entanto, ainda que diversos benefícios possam ser elencados com o uso de SOA para aplicações industriais, um desafio desta nova filosofia é como prover meios de interconexão entre dispositivos e aplicações que forneçam todos os requisitos exigidos por tais aplicações. Várias tecnologias e conceitos baseados na IoT encontraram seu caminho para a I4.0 e, especialmente, para a integração de dispositivos (Bangemann et al., 2014). Algumas das tecnologias desenvolvidas e mais utilizadas para aplicações de SOA industriais são o DPWS, REST e OPC UA (JAMMES et al., 2014; HENNEKE, ELATTAR & JASPERNEITE, 2015). Trabalhos recentes buscam a integração entre esses protocolos visando melhorar a interoperabilidade da SOA. COLOMBO et al. (2017) integra os padrões OPC UA e DPWS para uma aplicação de SOA na monitoração e controle de processos em plantas. Derhamy et al. (2017) apresenta um serviço de tradutor de protocolos, mapeando a comunicação dos protocolos CoAP, HTTP e MQTT no padrão OPC UA.

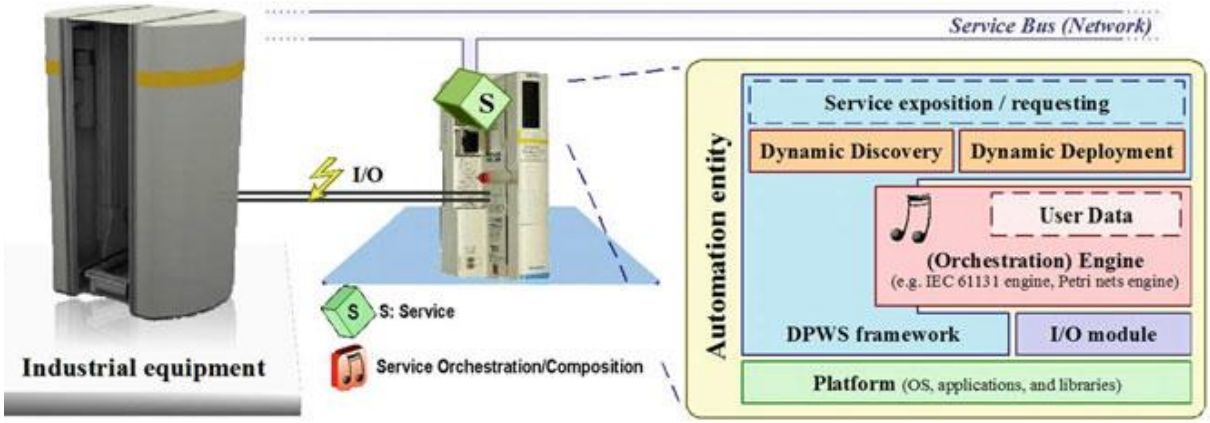
Na última década, diversos trabalhos desenvolveram SOA e avaliaram sua utilização em aplicações industriais e no contexto de CPS, IIoT e I4.0. Leitão et al. (2016) apresentam uma detalhada revisão de algumas iniciativas financiadas pela União Europeia, que são os projetos SOCRADES, IMC-AESOP, GRACE e ARUM. No trabalho são apresentadas as arquiteturas SOA desenvolvidas e discutidas implementações e casos de estudo onde estas arquiteturas

foram testadas e validadas. Nestes projetos é possível verificar a aplicação de SOA em todas as camadas tradicionais da ISA-95 (Figura 21), desde o nível de Dispositivos (SOCRADES), passando pelo nível de Controle e Supervisão (IMC-AESOP), até nível de Sistemas Corporativos (GRACE e ARUM). Outros projetos recentes, como o ARROWHEAD e o PRODUCTIVE 4.0 financiados pela União Europeia sobre desenvolvimento e utilização de SOA em aplicações industriais derivaram destes projetos citados anteriormente e demonstram a relevância atual desta temática para alcançar os novos objetivos requeridos pela Indústria 4.0.

O projeto SIRENA (*Service Infrastructure for Real-time Embedded Networked Applications*) (SIRENA, 2006), de 2003 a 2006, foi a primeira iniciativa focada no estudo de uma SOA para aplicações industriais. O objetivo do projeto foi desenvolver um *framework* para que dispositivos se comunicassem usando XML e *Web Services*, permitindo disponibilizar as funcionalidades de uma rede de dispositivos. Para isso foi desenvolvido o padrão DPWS, o qual foi a primeira tecnologia de *Web Service* baseado no padrão SOAP desenvolvida para o nível de dispositivos (JAMMES & SMIT, 2005). O projeto SODA (*Service Oriented Device & delivery Architectures*) SODA (2007), de 2006 a 2008, teve como objetivo a proposição de um modelo de arquitetura SOA baseada em *Web Service* que permitisse a interação entre dispositivos, usando o *framework* do projeto SIRENA e a tecnologia DPWS.

Os resultados dos projetos SIRENA e SODA foram base para os projetos SOCRADES (SOCRADES, 2008) e IMC-AESOP (IMC-AESOP Project, 2011) também apoiados pela União Europeia. O projeto SOCRADES (*Service Oriented Cross-layer infRAstructure for Distributed smart Embedded deviceS*) (SOCRADES, 2008), de 2006 a 2009) objetivou o desenvolvimento de uma plataforma de projeto, operação e gerenciamento de aplicações industriais baseado no conceito de SOA aplicado tanto no nível de dispositivos quanto no de aplicação. A Figura 22 apresenta a estrutura de um componente SOA padronizado no projeto.

Figura 22 – Estrutura de um Componente SOA no Projeto SOCRADES

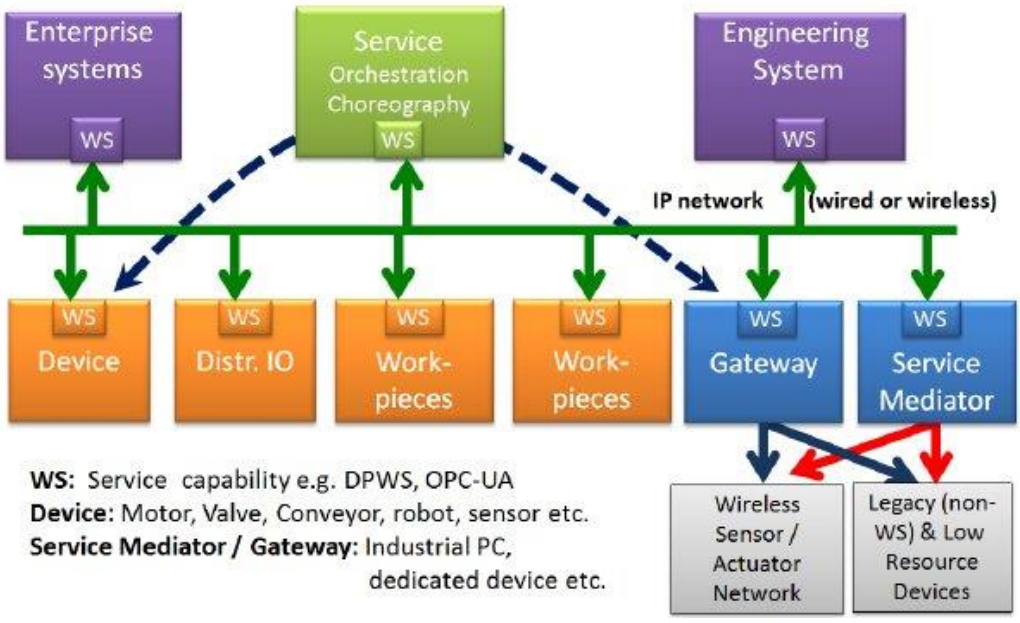


Fonte: SOCRADES (2008).

Neste projeto um ecossistema SOA foi proposto, adicionando aos resultados dos projetos anteriores alguns desenvolvimentos importantes como o de componente SOA, tipos de serviços diferentes e mecanismos para composição e orquestração e serviços disponíveis. De acordo com a Figura 22, um componente é o responsável pela integração de um equipamento industrial ou sistema na SOA. Esse componente é composto por um dispositivo embarcado inteligente, capaz de interagir com o ambiente físico (equipamento) e/ou com uma aplicação, cujas funcionalidades estão encapsuladas como *Web Services* e são oferecidas aos outros componentes da arquitetura através de um barramento de serviços (*Service Bus*) (COLOMBO, KARNOUSKOS & BANGEMANN, 2014). Essa configuração fornece, portanto, uma rede de componentes que é a base da SOA desenvolvida no projeto SOCRADES, conforme mostrado na Figura 23.

Na Figura 23 é possível visualizar uma estrutura na qual diversos componentes SOA, representados pelos quadrados de cores diferentes, podem ser interconectados através de *Web Services* (WS). Neste projeto diferentes tipos de serviços foram propostos para realização de atividades diferentes (CANNATA, GEROSA & TAISCH, 2008). Os serviços de dispositivos são representados pelos quadrados na cor laranja e correspondem à integração na arquitetura SOA de equipamentos, máquinas, interfaces de I/O e etc. Os serviços de interconexão (*Gateway* e *Mediator*) são representados pelos quadrados de cor azul e realizam a ponte (*middleware*)com a arquitetura SOA de equipamentos de diferentes redes industriais tradicionais (*Gateway*) e de equipamentos industriais legados que não possuem a capacidade de comunicação via Web Service (*Mediator*).

Figura 23 –SOA do Projeto SOCRADES

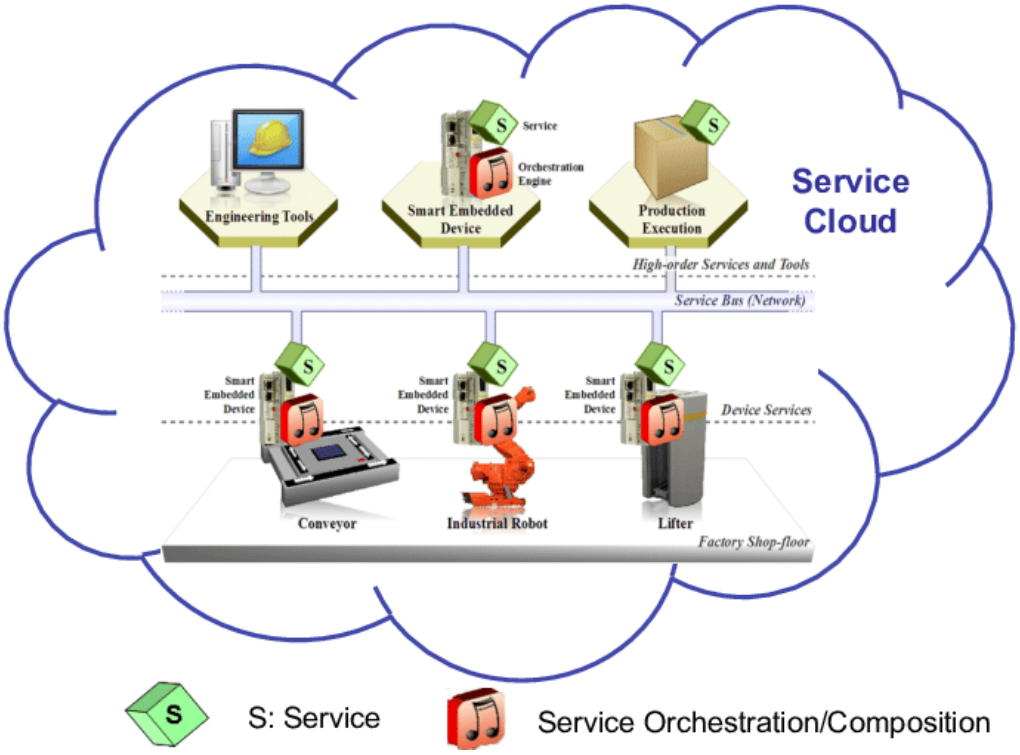


Fonte: SOCRADES (2008).

O projeto SOCRADES permitiu ainda a criação de serviços de aplicação, ou serviços relacionados à níveis hierárquicos mais altos na estrutura industrial, como Engenharia e sistemas corporativos. Estes serviços são representados pelos quadrados roxos. No entanto, a contribuição mais importante deste projeto foi viabilizar a criação de novos serviços e aplicações a partir da composição de diversos serviços alocados em diferentes dispositivos industriais. Para isso, um mecanismo de orquestração, representado pelo quadrado verde na Figura 23, foi desenvolvido usando o formalismo de Redes de Petri (MENDES et al., 2009).

O projeto IMC-AESOP (*ArchitecturE for Service-Oriented Process*) (IMC-AESOP, 2011), de 2010 a 2013, objetivou o desenvolvimento de uma 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*). A SOA em nuvem do projeto IMC-AESOP é mostrada na Figura 24.

Figura 24 – Arquitetura SOA em Nuvem do Projeto IMC-AESOP



Fonte: IMC-AESOP (2011).

O grande avanço da SOA do projeto IMC-AESOP foi a disponibilização dos serviços criados em uma infraestrutura de Computação em Nuvem (KARNOUSKOS et al., 2014). Este fato forneceu a arquitetura, principalmente, maior disponibilidade e capacidade de se adaptar às variações de uso da arquitetura devido à replicação de serviços em nuvem. Diversos serviços foram desenvolvidos para suportar todas as funcionalidades propostas para a arquitetura como

serviços de monitoramento, de controle, de alarmes, de simulação e de supervisão. O trabalho de Karnouskos et al. (2014) descreve detalhadamente a concepção dos serviços e o mecanismo de composição de serviços para obtenção de outras funcionalidades.

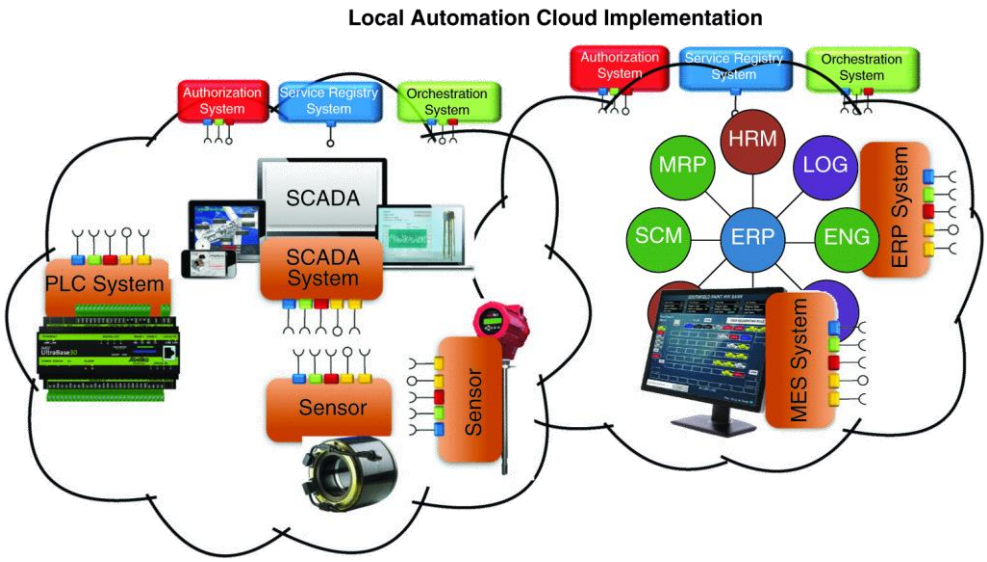
É importante verificar que todo o desenvolvimento dos projetos anteriores citados foi baseado na comunicação via *Web Service* entre os serviços através do padrão DPWS. Apesar de o DPWS ter sido amplamente utilizado e prover uma série de funcionalidades importantes para aplicações de SOA (COLOMBO, KARNOUSKOS & BANGEMANN, 2014; KARNOUSKOS et al., 2014), como recursos de descoberta de dispositivos, padronização e segurança na comunicação, o uso do padrão apresentava algumas desvantagens.

Como o DPWS é baseado no SOAP, é conhecido que o desempenho da comunicação de soluções que usam esse padrão é inferior devido à verbosidade do protocolo, ou seja, o ato de usar mais dados para transmitir a mesma informação (PAUTASSO, ZIMMERMANN & LEYMANN, 2008). Devido à estrutura da mensagem nesse padrão, a transmissão de poucos dados ou *bytes*, como é comum nas aplicações industriais, requer a adição de cabeçalhos que aumentam significativamente a quantidade de dados transmitidos (FELDHORST et al., 2009). Devido a esses fatores, entre outros, aplicações recentes têm focado no uso de outros padrões como o REST e o OPC UA em SOA, de forma a reduzir o impacto da publicação de serviços e da comunicação de mensagens no desempenho da arquitetura como um todo.

Espí-Beltran, Gilart-Iglesias & Ruiz-Fernandez (2017) utilizam o padrão REST para desenvolvimento de uma SOA. Os resultados deste trabalho, obtidos por meio da implementação e teste de uma SOA usando os protocolos CoAP e HTTP compatíveis com o REST, demonstram que o desempenho da arquitetura é adequado em face dos requisitos de comunicação (tempo de resposta, de transmissão e *jitter*) industriais atuais. Blomstedt et al. (2014) e Delsing (2017) apresentam o projeto *Arrowhead* que desenvolveu um *framework* para SOA que inclui um conjunto de serviços para implantação e documentação da arquitetura. Neste projeto também foi utilizado o protocolo CoAP para comunicação entre serviços.

Os resultados dos projetos SOCRADES e IMC-AESOP foram base para o desenvolvimento do projeto ARROWHEAD (ARROWHEAD, 2014), de 2013 a 2017. Estre projeto desenvolveu o conceito de nuvem de automação local (DELSING, 2017), no qual os componentes e dispositivos de automação eram disponibilizados como serviços, provendo interoperabilidade total para aplicações industriais, conforme apresentado na Figura 25. Neste projeto, toda a infraestrutura de integração e comunicação era realizada através de uma SOA, estruturada no formato de um *framework*. O *framework* Arrowhead atualmente é usado nos projetos *Arrowhead Tools* (ARROWHEAD TOOLS, 2019), com vigência de 2019 a 2022, e *Productive 4.0* (PRODUCTIVE 4.0, 2019), com vigência de 2017 a 2020.

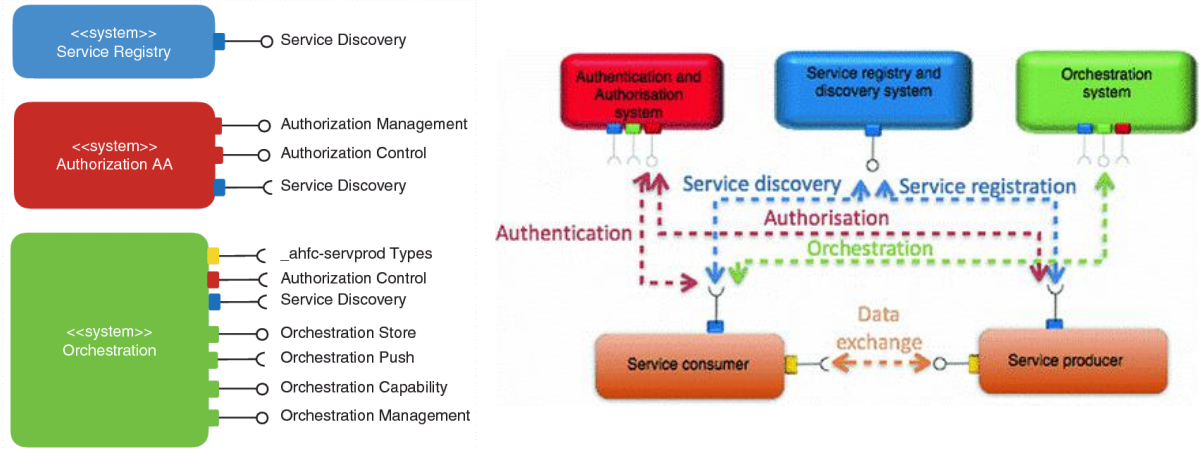
Figura 25 – Conceito de Nuvem Local na Arquitetura SOA do Framework Arrowhead



Fonte: Delsing (2017)

No *framework* Arrowhead, três conjuntos de serviços são requeridos para a implementação de qualquer aplicação: serviço de Registro, serviço de Autorização e o serviço de Orquestração. O serviço de Registro contem a listagem dos serviços disponíveis na arquitetura e permite a qualquer aplicação descobrir e consumir qualquer serviço disponível. O serviço de Autorização habilita o provedor de um determinado serviço a determinar e aceitar pela requisição de alguma aplicação interessada em consumir aquele serviço. E o serviço de Orquestração é o responsável por controlar a comunicação, de forma que a aplicação consumidora possa efetivamente usar o serviço desejado. Uma aplicação é a responsável por definir uma sequência ou composição de serviços de que devem ser consumidos para execução de uma determinada tarefa (Paniagua, Eliasson & Delsing, 2020). A Figura 28 apresenta um exemplo dessa interação entre serviços no *framework* Arrowhead.

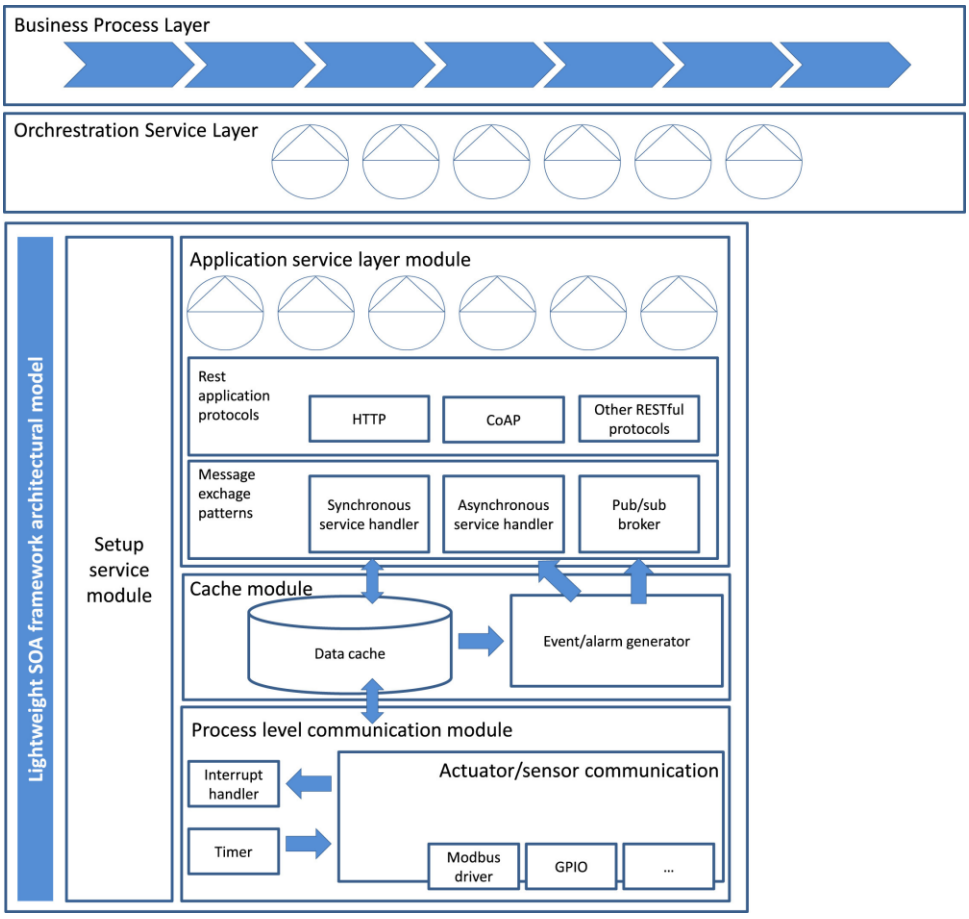
Figura 26 – Interação entre Serviços na Arquitetura SOA do Framework Arrowhead



Fonte: Delsing (2017)

Uma visão geral de uma SOA usando REST é apresentada na Figura 27. Espí-Beltran, Gilart-Iglesias & Ruiz-Fernandez (2017) afirmam que a arquitetura pode ser adaptada para qualquer aplicação industrial por meio do uso de um serviço de parametrização e configuração (*Setup Service*). A arquitetura no seu nível inferior permite a comunicação com dispositivos e equipamentos legados por meio 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 por meio de comunicação via protocolos compatíveis com o padrão REST. Esses mecanismos de comunicação podem ser do tipo síncronos (*request/response*) ou assíncronos (*publish/subscribe*). Acima desses serviços disponibilizados existem outras duas camadas: Camada de Orquestração de Serviços, responsável pela coordenação na execução dos serviços, e a Camada de Processo de Negócios, responsável pela criação de aplicações do usuário por meio de composição de serviços.

Figura 27 – Arquitetura SOA-REST para Aplicações Industriais



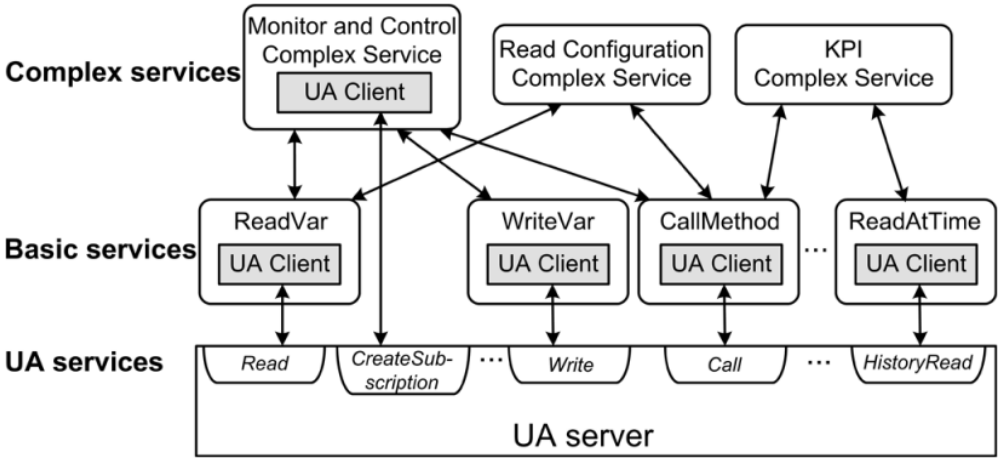
Fonte: Espí-Beltran, Gilart-Iglesias & Ruiz-Fernandez (2017)

Outra tecnologia de comunicação que tem sido bastante usada em aplicações de CPS, IIoT e I4.0 é o protocolo OPC UA (WOLLSCHLAEGER; SAUTER; JASPERNEITE, 2017). Por se tratar de um protocolo orientado a serviços, o OPC UA inerentemente suporta aplicações

de SOA (LEITNER & MAHNKE, 2006). A OPC Foundation fornece por meio da especificação de número 4 (*Services*), um total de dez conjuntos de serviços básicos para o OPC UA (OPC FOUNDATION, 2018). A requisição de serviço e a resposta são transmitidas por meio da troca de mensagens entre clientes e servidores OPC UA. O padrão OPC UA fornece diversas opções de protocolos na camada de transporte como SOAP/HTTP, TCP/IP binário e *publish/subscribe* com os protocolos MQTT e AMQP. Além da variedade de mecanismos de comunicação, outras vantagens para o uso do OPC UA em SOA são os mecanismos de segurança e a estrutura de dados padronizada.

Girbea et al. (2014) apresentam uma SOA baseada no OPC UA para planejamento e operação de sistemas produtivos industriais. A arquitetura consiste de três componentes principais que são: um conjunto de servidores OPC UA responsáveis pela integração dos dispositivos, um conjunto de serviços responsáveis pelas funcionalidades requeridas para a aplicação e um planejador de ações, responsável pela computação e sequenciamento da execução de serviços para realização da ação requerida. A Figura 28 apresenta um diagrama do relacionamento entre os serviços disponíveis na arquitetura. Como pode ser verificado, os serviços são organizados em duas camadas (*Basic* e *Complex Services*), sendo que os serviços básicos são os fornecidos pelo padrão OPC UA e os serviços complexos são os serviços criados a partir da composição dos serviços básicos.

Figura 28 – Relacionamento entre Serviços numa Arquitetura SOA-OPC UA para Aplicações Industriais



Fonte: Espí-Beltran, Gilart-Iglesias & Ruiz-Fernandez (2017)

Um outro exemplo de composição de serviços OPC UA para aplicações industriais é apresentado em Theorin, Hagsund & Johnsson (2014). Neste trabalho, o OPC UA é integrado à ferramenta *JGrafchart*, permitindo a composição de serviços a partir de um ambiente gráfico com suporte à linguagem de programação de controladores SFC (*Sequential Function Chart*) da norma IEC 61131-3.

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).

Dois projetos europeus estão focando no desenvolvimento de MOA para aplicações no contexto de CPS, IIoT e I4.0. Ciavotta et al. (2017) apresentam uma proposta de MOA para fomentar a implantação do conceito de fábrica digital. Esta arquitetura faz parte do projeto MAYA (*Multi-disciplinArY integrated simulAtion and forecasting tools)* (MAYA, 2018), cujo foco é o desenvolvimento de aplicações de CPS e I4.0 e possui como diferencial o suporte para a simulação de processos industriais e criação de gêmeos digitais (*Digital Twin*). Na arquitetura são propostos cinco grupos principais de serviços que se comunicam via padrão REST com HTTP e *WebSocket* via TCP/IP. Um destaque pode ser dado aos microserviços Orquestrador (*Orchestrator*) e Agendador (*Scheduler*), os quais coordenam e organizam os outros serviços para permitir a composição e criação de serviços de alto nível e aplicações de processo. Innerbichler et al. (2017) apresentam uma proposta de MOA em nuvem para construção de uma plataforma colaborativa para a I4.0, que permita o monitoramento, a otimização e a negociação em tempo real com base em IoT nas cadeias de fornecimento (*Supply Chain*) de manufatura. Esta arquitetura faz parte do projeto NIMBLE (NIMBLE, 2018).

Os levantamentos realizados demonstram a importância da pesquisa e desenvolvimento de SOA, e mais recentemente MOA, para fornecer os novos requisitos das aplicações 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 e dificuldades deste tipo de aplicação.

3.2 SÍNTESE DO CAPÍTULO

Este capítulo apresentou uma revisão sobre as principais SOA desenvolvidas para aplicações industriais. A revisão realizada demonstrou uma grande quantidade de desenvolvimentos de SOA em aplicações industriais diversas e usando diferentes tecnologias de comunicação e integração. No entanto, ainda há uma carência relevante de desenvolvimentos

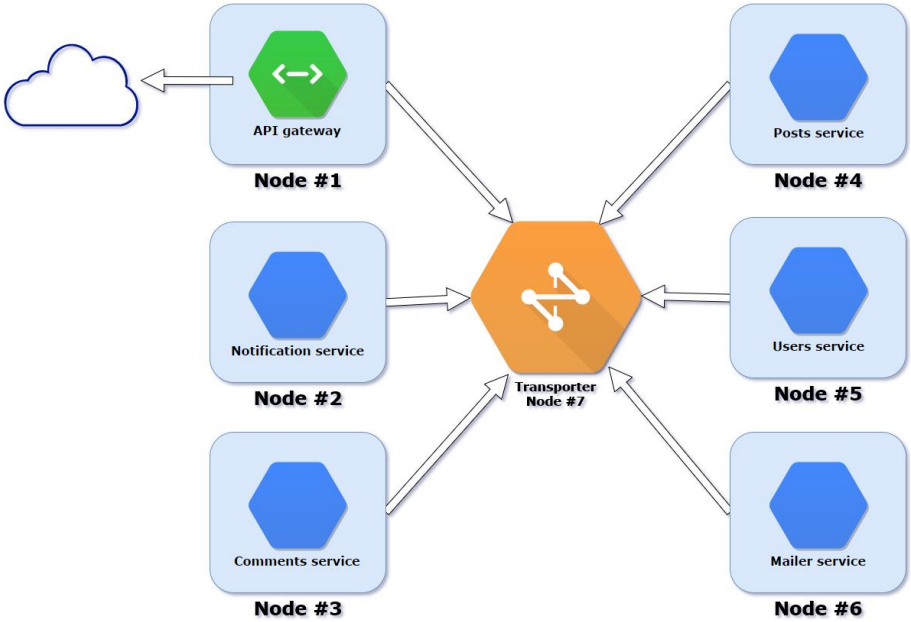
de arquiteturas orientadas a microserviços para aplicações industriais de IIoT e I4.0, principalmente em relação há uma análise mais detalhada de desempenho e vantagem de aplicação. Baseando-se nessa demanda, esta pesquisa utilizou as arquiteturas discutidas nesta seção, as quais foram utilizadas como referência para a proposição e desenvolvimento da MOA para aplicações de IIoT e I4.0 desse trabalho.

4 FRAMEWORK MOLECULER

Um *framework* representa uma estrutura base ou uma plataforma de desenvolvimento, como um arcabouço, que contém ferramentas, bibliotecas, sistemas e componentes que agilizam o processo de desenvolvimento de uma solução específica. O *framework Moleculer* é uma estrutura de desenvolvimento com a linguagem JavaScript para desenvolvimento de aplicações orientadas a microserviços (MOLECULER, 2018). O *framework* é de código aberto (*open source*) e roda sobre a plataforma Node.js.

A Figura 29 apresenta a MOA do *framework Moleculer*, na qual os serviços são executados em nós individuais que se comunicam via protocolo de comunicação (*transporter*) por meio de um *broker* de mensagens. Nesta arquitetura distribuída, a latência da comunicação entre os serviços não é insignificante, mas os serviços podem ser replicados para proporcionar resiliência e tolerância a falhas.

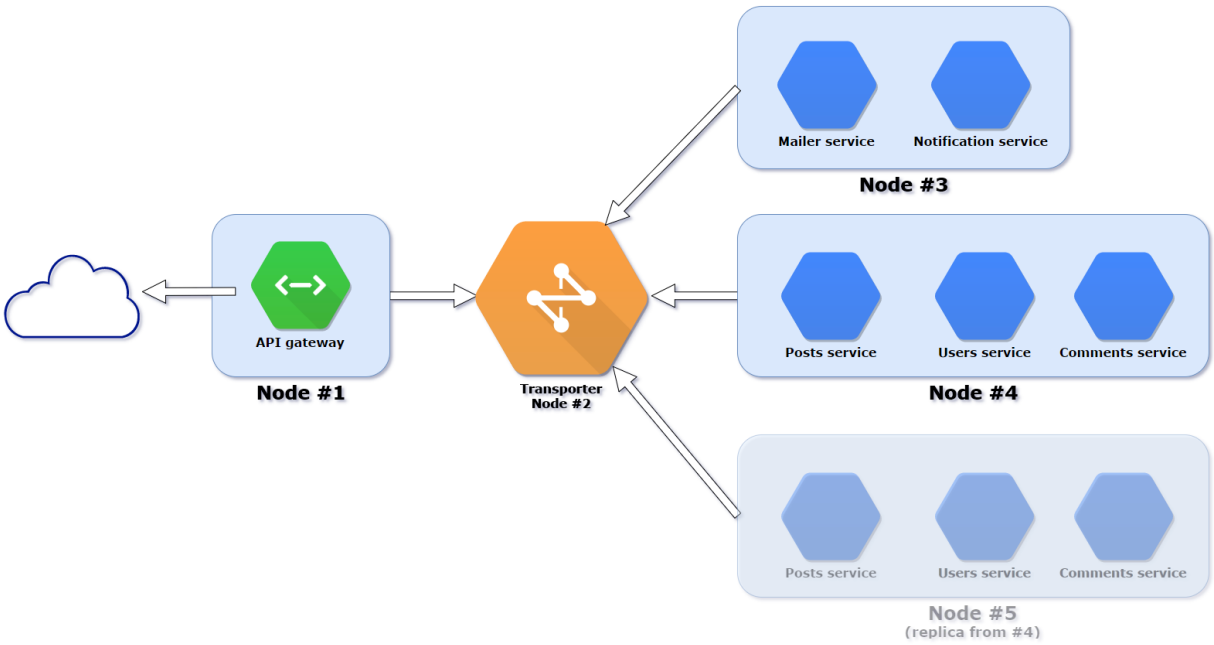
Figura 29 - Arquitetura Orientada a Microserviços do *Moleculer*



Fonte: Moleculer (2018)

Outras estruturas de interconexão mista entre os serviços também são permitidas no *Moleculer*, permitindo a melhor configuração para cada aplicação. Um exemplo é apresentado na Figura 30, na qual é possível visualizar o agrupamento de vários serviços num mesmo nó numa arquitetura mista. Nessa configuração não há necessidade de comunicação via *transporter* entre esses serviços alocados no mesmo nó, de forma que a latência é reduzida. Caso um nó ou serviço esteja sobrecarregado, é possível realizar a replicação destes conforme mostrado (Node 5).

Figura 30 – Exemplo de Interconexão Mista de Serviços do *Moleculer*



Fonte: Moleculer (2018)

Na arquitetura do *Moleculer*, todos os microserviços são iguais, não havendo nenhum tipo de hierarquia ou prioridade. Uma grande vantagem desse *framework* é que todos os microserviços desenvolvidos possuem disponível um recurso automático de registro e descoberta. Dessa forma, todos os serviços existentes são informados após a criação de um novo serviço ou 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 claro na arquitetura (Figura 29 e Figura 30) representam os nós de gerenciamento do *Moleculer* para suporte ao desenvolvimento, descoberta e configuração dos (micro) serviços. Cada nó pode conter um ou mais serviços e é responsável pelo gerenciamento dos seus dados, possuindo seu próprio banco de dados. Cada serviço disponibilizado na arquitetura é representado com um hexágono azul. 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 29 e na Figura 30 cada serviço disponibilizado na arquitetura é representado com um hexágono azul escuro.

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 por meio de chamadas via rede ou nuvem por meio do protocolo 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 e numa infraestrutura para a mensageria, 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 mensageria, ou seja, para se comunicar com os outros microserviços. O grande benefício dessa abordagem é o desacoplamento entre produtores e consumidores de eventos e dados. 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* de mensageria é uma técnica que visa solucionar a comunicação entre sistemas completamente diferentes de uma maneira confiável. *Broker* de mensagem faz com que seja possível integrar tais sistemas para 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* é um estilo arquitetônico em que as aplicações são decompostas em serviços de baixo acoplamento, oferecendo modularidade, tornando as aplicações mais fáceis de desenvolver, testar, implantar e, o mais importante alterar e manter. As principais classes de componentes operacionais do *framework Moleculer* são:

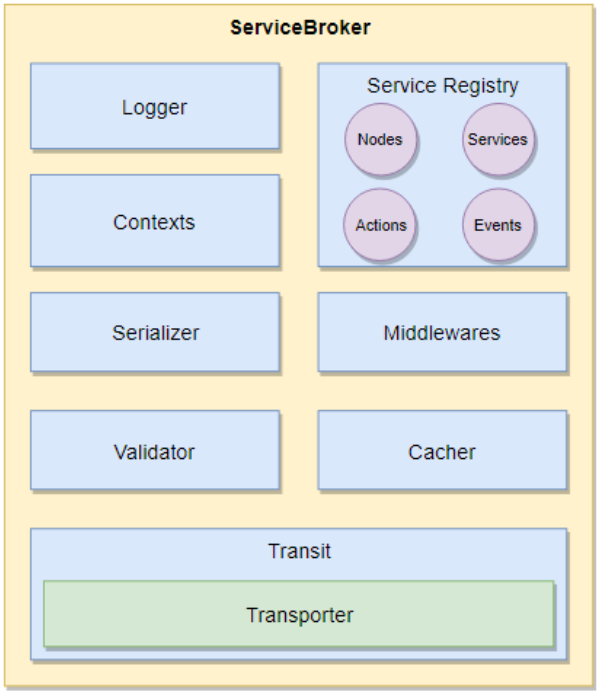
- ✓ *ServiceBroker*;
- ✓ *Service*.

4.1 INTERMEDIÁRIO DE SERVIÇO (SERVICE BROKER)

O *ServiceBroker* é o principal componente da estrutura do *Moleculer*. É um contêiner com recursos embarcados para simplificação do desenvolvimento e mediação dos microserviços, sendo responsável pela configuração dos nós como: nome do nó, registro de serviços, descoberta automática de serviços, *cache* de variáveis, *serializer*, *middlewares* e *Transporter*, entre outros recursos como pode ser visto na Figura 31.

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 32 e Figura 33.

Figura 31 – Contêiner *ServiceBroker* do *Moleculer*



Fonte: Moleculer (2018)

Figura 32 - Configuração Padrão *ServiceBroker*

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

Fonte: Moleculer (2018)

Figura 33 - Configuração Personalizada do *ServiceBroker*

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

Fonte: Moleculer (2018)

O objeto *ServiceBroker***,** mostrado na Figura 34, utiliza um serviço de mensagens (*Transporter***),** para se comunicar com outros nós. A Tabela 3 apresenta as explicações da configuração de cada comando do *ServiceBroker* da Figura 34.

Figura 34 - Configuração do *ServiceBroker* utilizando Transporter

```
const { ServiceBroker } = require("moleculer");
const broker = new ServiceBroker({
    nodeID: "node-1",
    transporter: "nats://localhost:4222",
    logLevel: "debug",
    requestTimeout: 5 * 1000,
    requestRetry: 3
});
```

Fonte: Moleculer (2018)

Tabela 3 - Descrição das Configurações no *ServiceBroker*

Opção	Descrição
nodeID	O nome node (nó) é “node-1”.
transporter	Configuração do *transporter* utilizado, que neste caso é “Nats” que está se comunicando os outros microserviços pela porta 4222.
logLevel	Nível de *log* para mensagens de console dos *logs* de informação do funcionamento do microserviço, como (*trace, debug, info, warn, error, fatal*). Que neste caso é “debug”
requestTimeout	Tempo em milissegundos de espera de uma resposta a uma requisição.
requestRetry	Número de tentativas de solicitação de requisição de comunicação com um microserviços.

4.2 SERVIÇO (SERVICE)

O componente *Service* representa um microserviço no *framework Moleculer*. Para criar um microserviço, deve-se definir um esquema (*schema*) com as seguintes propriedades:

- ✓ ***name***: nome do microserviço. É a primeira parte do nome da ação de quando é chamado o microserviço.
- ✓ ***version*:** É usada para executar várias versões do mesmo microserviço, com ações diferentes.
- ✓ ***settings****:* configurações do microserviço, como porta de comunicação. Essas configurações são enviadas durante o procedimento de descoberta do serviço;
- ✓ ***actions****:* ações ou funções que compõem o funcionamento do microserviços, podendo ser chamadas internamente por outros microserviços ou externamente via API REST;
- ✓ ***methods****:* métodos são funções privadas do microserviço que somente são executadas internamente;
- ✓ ***events****:* eventos podem ser disparados conforme execução das ações do microserviço. Cada microserviço publica um evento sempre que necessário e outros microserviços podem se inscrever em eventos publicados.

Na Figura 35 é apresentado o objeto *broker* criado a partir da classe *ServiceBroker* que utiliza a função ***createServive*({*schema*}).** Na Tabela 4 é descrito o funcionamento de cada propriedade do ***schema*** (esquema) da função ***createService*()** apresentada na Figura 30.

Figura 35 - Criação de um Microserviço usando Função *createService*()

```
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");
        }
    }
});
```

Fonte: Moleculer (2018)

Tabela 4 - Descrição do Esquema das Propriedades dos Microserviços

Propriedade	Descrição
name	O *name* é uma propriedade obrigatória na criação do microserviço, portanto, deve ser definido. É nome do microserviço. Que neste caso o *name* é ***"posts".***
actions	A *actions* é a propriedade para execução de uma ação do microserviço podendo ter uma ou mais ações. O nome da ação é ***get*** que envia uma mensagem para o console "***Log message via Service logger*** ". Que neste caso *actions* é ***"get"*** .

4.2.1 Versão (Version)

A propriedade *Version* é usada para executar várias versões do mesmo microserviço, com ações diferentes. Na Figura 36 é apresentado um exemplo de código de microserviço "Versão 1", que não tem nenhuma ação, e na Figura 37 a "versão 2", que tem uma ação com o nome **find()**.

Figura 36 – Exemplo da Propriedade *Version* de um Microserviço no *Moleculer*

```
{
    name: "posts",
    version: 1
}
```

Fonte: Moleculer (2018)

Figura 37 - Exemplo da Propriedade *Version* de um Microserviço no *Moleculer* com Acão

```
{
    name: "posts",
    version: 2,
    actions: {
        find() {...}
    }
}
```

Fonte: Moleculer (2018)

4.2.2 Configurações (Settings)

A propriedade *Settings*, Figura 38, tem a função de configuração dos recursos disponíveis nos microserviços. Por exemplo, a configuração da porta de comunicação dos comandos API REST que serão enviados por meio do microserviço Gateway.

Figura 38 – Exemplo da Propriedade *Settings* de um Microserviço no *Moleculer*

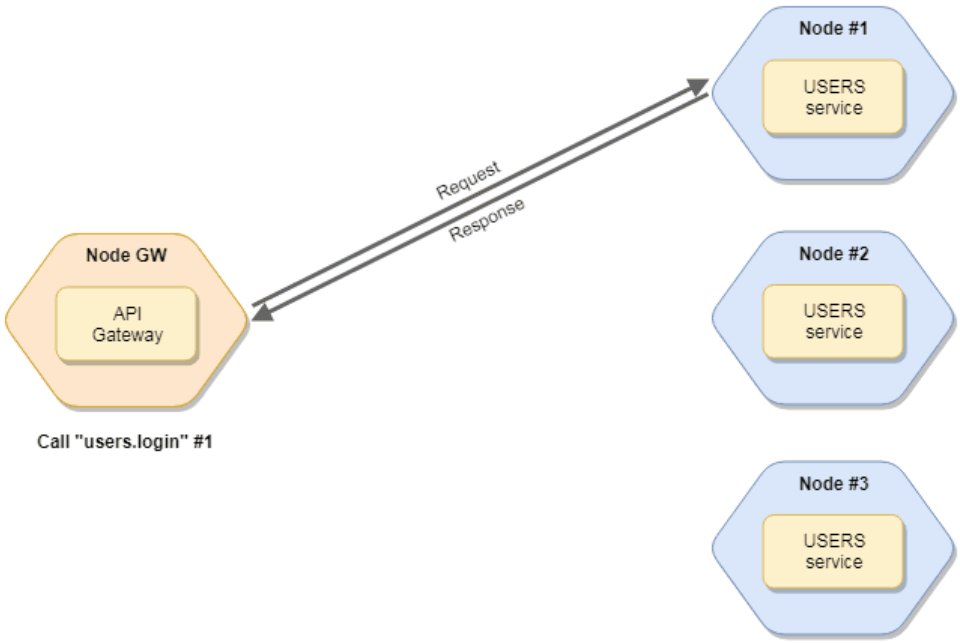
```
const ApiGwService = require("moleculer-web");

module.exports = {
    name: "api",
    mixins: [ApiGwService]
    settings: {
        // Change port setting
        port: 8080
    },
    actions: {
        myAction() {
            // Add a new action to apiGwService service
        }
    }
}
```

Fonte: Moleculer (2018)

4.2.3 Ações (Action)

A propriedade *Action* pode ter dois tipos de chamadas. As chamadas internas que são executadas pela função ***broker.call*** ou externas via API REST por meio do microserviço *Gateway*. As chamadas externas das ações utilizam métodos públicos que podem ser chamados por qualquer objeto programado no microserviço. O método utilizado para as chamadas das ações é RPC, que são solicitadas via requisições HTTP. O diagrama da Figura 39 apresenta o método de comunicação por RPC (*Remote Procedure Call*) do *Moleculer*.

Figura 39- Método de Comunicação RPC das Chamadas Externas de Ações



Fonte: Moleculer (2018)

Essas chamadas externas das ações são executadas via API REST por meio do microserviço Gateway, por exemplo, usando o comando GET “http://localhost:3000/get” e o resultado apresentado via console pode ser verificado na Figura 40. RPC são chamadas remotas de procedimento, que são 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 as de procedimentos locais.

RPC é uma tecnologia popular para a implementação do modelo cliente-servidor de computação distribuída. 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 tiver várias instâncias de serviços, o *ServiceBroker* fará o balanceamento de carga da solicitação entre as instâncias. O método de balanceamento automático de carga tem função de distribuir dinamicamente a carga da comunicação entre os microserviços uniformemente.

Figura 40- Exemplo de Chamada da Acão get via *broker.call* de um Microserviço do *Moleculer*

```
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)

Para chamar uma ação interna, é usado o método ***broker.call***. Quando o método ***broker.call*** é chamado, o *ServiceBroker* procura o microserviço em um nó que tem uma determinada ação e o chama. A função retorna um *Promise*. Em JavaScript *Promise* é um objeto

usado para processamento assíncrono. Um *Promise* (de "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.

Na Figura 40 é apresentada a função ***action*** do microserviço, que neste caso é chamado de ***get***. Para chamada interna dessa função, é preciso iniciar o microserviço via comando ***broker.start()*** e depois chamar **ação** via comando ***broker.call*.**

4.2.4 Métodos (Methods)

A propriedade *Methods* (métodos) é função privada 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 41, e serem chamadas a partir das funções de ação ***action***. As funções privadas, não podem ser chamadas via ***broker.call***. Para executar chamadas internas dos manipuladores de ações e de eventos, deve-se realizar por meio da função ***methods.***

Figura 41- 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)

4.2.5 Eventos (Events)

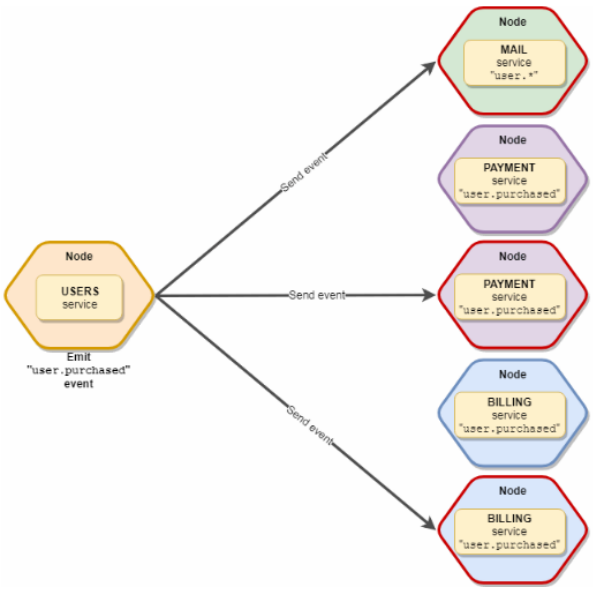
O *ServiceBroker* possui um barramento integrado para suportar uma arquitetura orientada a eventos. Quando ocorrer um evento, pode ser enviado uma mensagem para os nós locais e remotos. A propriedade *events* (eventos) consiste de produtores de eventos que geram um fluxo de dados, e consumidores dos eventos que os escutam. Os eventos são entregues após a execução da propriedade, para que os consumidores possam responder imediatamente conforme os eventos ocorram. Os produtores são separados dos consumidores. Um produtor não sabe quais consumidores estão ouvindo. Os consumidores também são separados uns dos outros e cada consumidor vê todos os eventos. Isso difere do padrão de Consumidores Concorrentes, onde os consumidores removem mensagens de uma fila e uma mensagem é processada apenas uma vez.

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

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

No ***Balanced events*** (eventos balanceados), os microserviços são organizados em grupos lógicos ouvintes dos eventos. Isso significa que apenas um microserviço ouvinte é acionado em cada grupo, como pode observado na Figura 42.

Figura 42- Diagrama de Evento Balanceado para Microserviço no *Moleculer*



Fonte: Moleculer (2018)

O nome do ***group*** (grupo) vem do nome do microserviço, mas pode ser definido na programação das propriedades do evento como pode ser visto na Figura 43.

Figura 43 – Exemplo da Propriedade Nome 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 usado 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 44.

Figura 44 – 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)

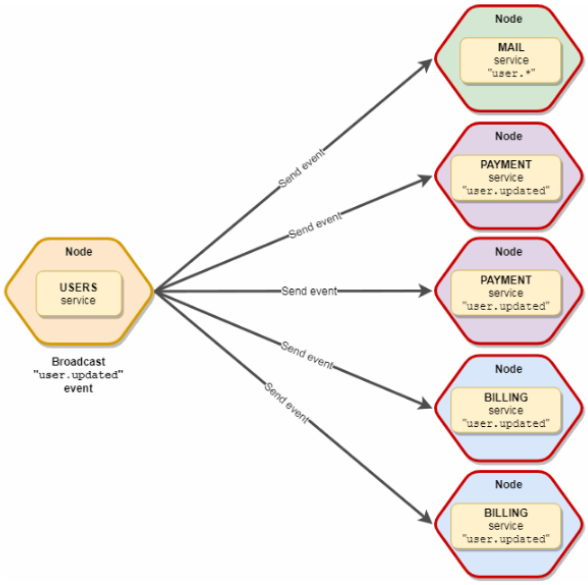
No ***Broadcast event,*** a transmissão dos eventos é enviada para todos os serviços locais e remotos disponíveis. Na Figura 45 é apresentado o método ***broker.broadcast*** para transmissão de eventos para todos os microserviços. A transmissão dos eventos não é balanceada, sendo que todas as instâncias de serviço recebem a mensagem do evento ocorrido como pode ser vista na Figura 46.

Figura 45 – Exemplo do Método *broker.broadcast* de um Microserviço do *Moleculer*

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

Fonte: Moleculer (2018)

Figura 46- 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).

4.3 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:

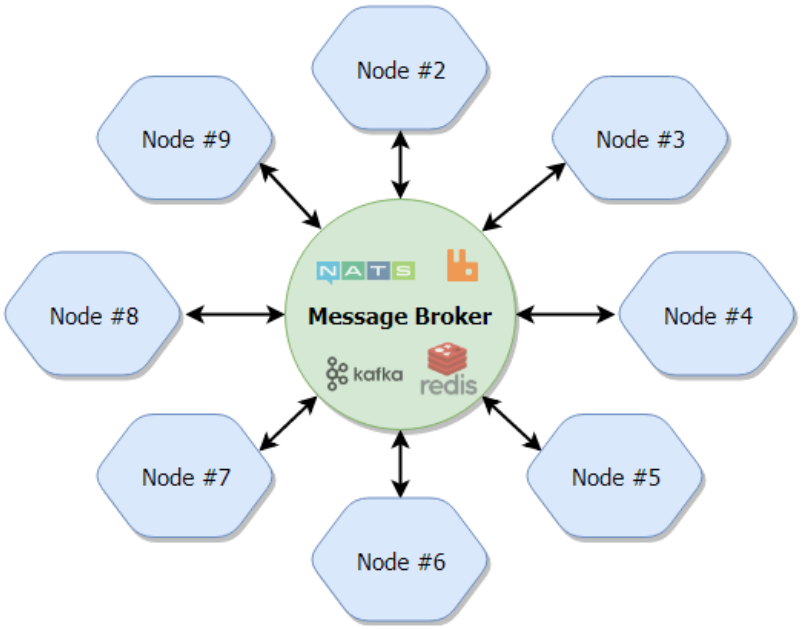
- ✓ Chamada da ação *hello* do microserviço teste: http://IP:3000/test/hello
- ✓ Chamada da ação *add* com parâmetros do microserviço *math*: http://IP:3000/math/add?a=25&b=13
- ✓ Chamada de informações do nó (microserviço) *health*: http://IP:3000/~node/health.
- ✓ Chamada de listagem das ações de todos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

4.4 TRANSPORTER

O serviço *Transporter* é responsável pela de troca de mensagens entre os microserviços da arquitetura MOA, conforme é apresentado na Figura 47. O *framework Moleculer*

disponibiliza uma série de mecanismos diferentes para comunicação entre os microserviços como: TCP/IP, NATS, MQTT, AMQP e etc.

Figura 47 - 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*.

5 ARQUITETURA ORIENTADA A MICROSERVIÇOS (MOA)

Esta pesquisa focou no estudo e desenvolvimento de uma arquitetura orientada a microserviços (MOA) para aplicações de automação e controle de processos com foco na I4.0. Visando agilizar, organizar e fornecer escalabilidade para o desenvolvimento da arquitetura proposta nesta pesquisa, o *framework Moleculer* foi utilizado como base. O *framework Moleculer* é uma estrutura de desenvolvimento com a linguagem *JavaScript* e plataforma Node.js para aplicações orientadas a microserviços. O uso desse *framework* mitigou a necessidade do desenvolvimento de um mecanismo de orientação a serviços para uso na arquitetura, que não era o foco deste trabalho. Além disso, proporcionou robustez e incorporou uma série de recursos e funcionalidades para a arquitetura MOA proposta.

Neste capítulo são apresentadas informações detalhadas sobre a proposta do trabalho da MOA com uso do *framework Moleculer*. Uma discussão sobre o conjunto de serviços definido para a arquitetura é apresentada, bem como sobre os tipos de composição de serviços que podem ser realizados para criação das tarefas e aplicações requeridas.

5.1 PROPOSTA DO TRABALHO E DESCRIÇÃO DA ARQUITETURA

A arquitetura MOA proposta com base no *framework Moleculer* pode ser vista na Figura 48. É importante verificar que a estrutura da arquitetura MOA foi proposta seguindo os componentes do *framework Moleculer*, mostrado na Figura 29, para facilitar a compreensão. A proposta discute o desenvolvimento de um conjunto de diferentes microserviços e aplicações para realização de atividades relacionadas à IIoT e I4.0. A arquitetura proposta busca suportar aplicações a partir da criação de serviços para: aquisição de dados (DAQ), monitoramento remoto e supervisão, otimização de processos (KPIs), programação e controle de processos e identificação de sistemas. No entanto, a arquitetura pode ser facilmente expandida, através da criação e composição de novos serviços, para permitir novas aplicações como virtualização, simulação de sistemas, segurança da informação e análise de dados usando Big Data e etc.

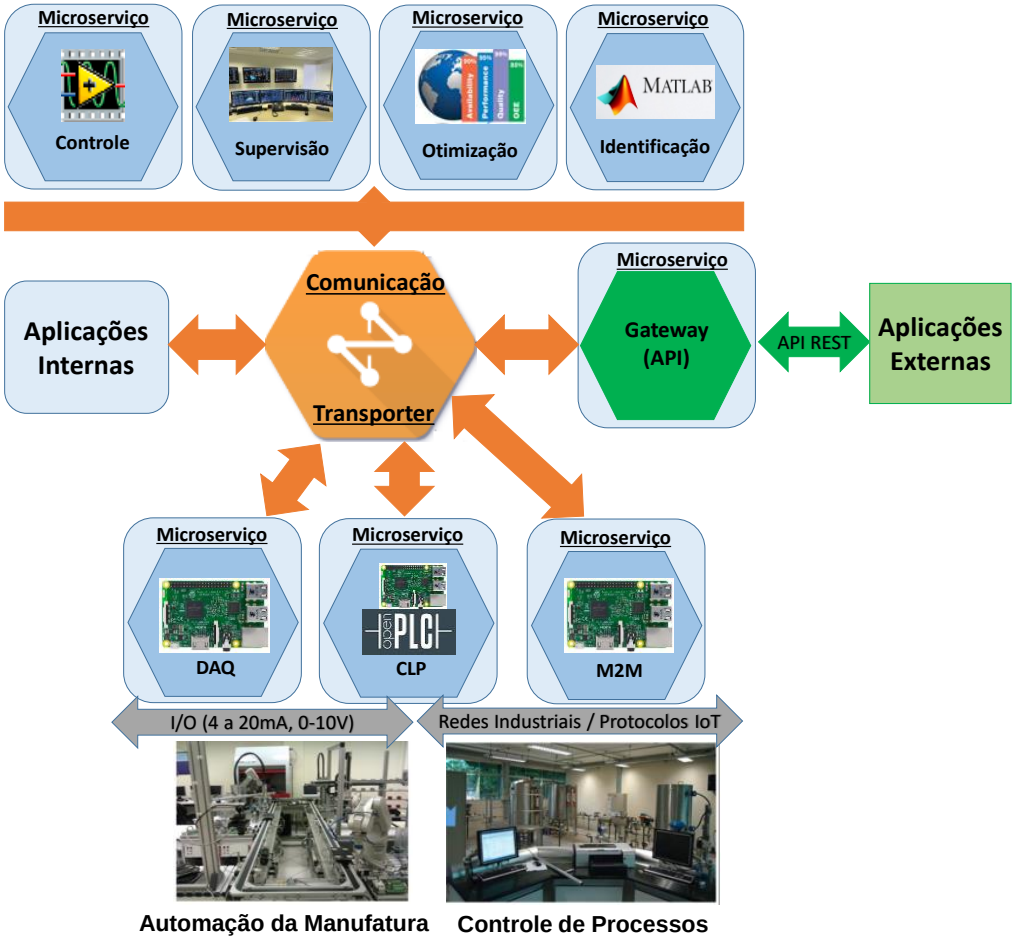
Os microserviços que compõe a MOA são divididos em dois níveis hierárquicos:

- ✓ Microserviços de Processos/Negócios;
- ✓ Microserviços de Infraestrutura.

Os Microserviços de Processos/Negócios executam as funcionalidades de alto nível como Controle, Supervisão, Monitoramento, Otimização e Identificação de Sistemas e geralmente necessitam operar em composição com outros serviços. Os Microserviços de Infraestrutura

como Comunicação (*Transporter*), Gateway, M2M, CLP e DAQ são os responsáveis por prover as funcionalidades básicas para composição dos serviços de alto nível das aplicações.

Figura 48 - Diagrama da MOA com Framework *Moleculer* para Aplicações de IIoT e I4.0



Fonte: Autor

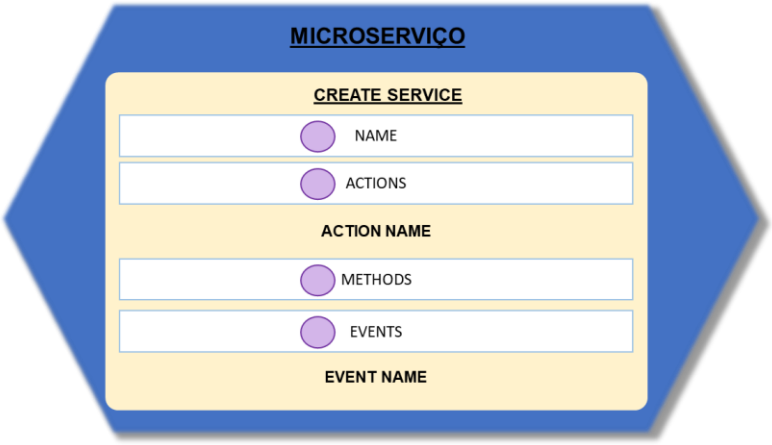
Os microserviços podem ser executados em três tipos de plataforma, de acordo com a aplicação e/ou necessidade: sistema embarcado representado por um computador de placa única (Raspberry Pi 3) rodando Linux, um computador (PC) com sistema operacional (Windows, Linux, MAC) e uma plataforma em nuvem comercial ou industrial (local). Serviços possíveis de operar em nuvem funcionariam numa estrutura de PaaS (*Platform as a Service*). Toda comunicação realizada pela arquitetura tem como base o protocolo Ethernet TCP (UDP)/IP, fornecendo grande interconectividade.

5.1.1 Descrição dos Serviços e Composição de Aplicações

Os serviços da MOA da Figura 48 oferecem uma funcionalidade principal, podendo executar diferentes ações, métodos e eventos relacionados àquela funcionalidade. Conforme

seção 4.2, um microserviço no *framework Moleculer* é definido por um esquema (*schema*) com as seguintes propriedades: *name*, *version*, *settings*, *actions*, *methods* e *events*. Visando padronizar a apresentação e facilitar a compreensão dos serviços propostos neste trabalho, o esquemático padrão da Figura 49 foi criado. Dessa forma, os microserviços são apresentados da mesma forma e suas funcionalidades discutidas no texto. Somente os serviços Gateway e *Transporter*, que são disponibilizados pelo *framework Moleculer*, não seguem esta estrutura.

Figura 49 – Esquemático Padrão de um Microserviço da Arquitetura MOA com Framework *Moleculer*



Fonte: Autor

Os microserviços no *Moleculer* executam ações por meio de "rotas". Para entender melhor o endereçamento das ações de um microserviço, é possível analisar o trecho de um código de um microserviço do *Moleculer*.

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

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

✓ *Localhost*:3000 supondo que esteja usando o servidor API Gateway num computador (*Localhost*). Se for em outro dispositivo, seria usado o IP do mesmo (IP: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, que 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>.

5.1.1.1 Microserviços de Infraestrutura

Microserviço DAQ

Este serviço é responsável por realizar a atividade de aquisição e atualização de dados de variáveis de interesse via *hardware* dedicado alocado no processo. Este serviço pode ser hospedado ou rodar em dois tipos de plataforma: computador (PC) ou sistema embarcado. Essa atividade de aquisição e atualização de dados é transparente aos outros microserviços ou aplicações que utilizam o microserviço DAQ, de forma que os mesmos não precisam se preocupar com a especificação do *hardware* e nem com o código (*software*) necessário para a aquisição/atualização dos dados das variáveis de interesse. Isso significa, por exemplo, que o microserviço DAQ poderia rodar numa plataforma baseado em computador (PC), e comunicar via USB ou PCI com alguma placa ou módulo DAQ para realização das atividades de aquisição/atualização das variáveis. Ou também poderia rodar num sistema embarcado (Raspberry Pi) e usar de *hardware* externo ou conexão de I/O direta para realização das atividades de aquisição/atualização das variáveis de interesse. Portanto, de forma resumida, para a aplicação que usa o microserviço DAQ, não importa qual *hardware* e *software* estão sendo usados para aquisição/atualização da variável, e sim somente a informação/dado da variável.

De acordo com a Figura 50, o microserviço DAQ possui basicamente ações e eventos. As ações estão relacionadas às funcionalidades principais: Aquisição de uma entrada (canal de I/O) e Atualização de uma saída (canal de I/O).

Figura 50 – Microserviço DAQ da MOA

Fonte: Autor

A aquisição de dados de uma entrada pode ser realizada de duas formas: Periódica, onde o microserviço periodicamente (conforme um período de *T* segundos) envia o dado da variável via *transporter* 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 citar 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.

Microserviço M2M

O microserviço M2M é responsável pela aquisição e atualização de dados das variáveis de interesse do processo via protocolo de comunicação com dispositivos/remotas/equipamentos alocados no processo. Este serviço pode ser hospedado ou rodar em dois tipos de plataforma: computador (PC) ou sistema embarcado. Essa atividade de aquisição e atualização de dados via comunicação M2M é transparente aos outros microserviços ou aplicações que utilizam o microserviço M2M, de forma que os mesmos não precisam se preocupar com qual tipo de *hardware*, protocolo de comunicação e nem com o código (*software*) necessário para a aquisição/atualização dos dados das variáveis de interesse. Portanto, o objetivo do serviço M2M é realizar uma ponte, como um *middleware*, entre a MOA do *Moleculer* e dispositivos/remotas/equipamentos industriais que suportam a comunicação em um dos protocolos disponíveis no microserviço M2M.

Os protocolos definidos para comunicação via microserviço M2M são baseados no protocolo IP com foco em conexões UDP ou TCP, sendo respectivamente o protocolo de aplicações de IoT CoAP (*Constrained Application Protocol*) e protocolo de Ethernet Industrial MODBUS TCP/IP. Esses protocolos são adequados para uso em *hardware* com baixa capacidade de memória e baixo consumo energético, como microcontroladores com baixa capacidade de processamento. É importante citar que embora a proposta para este serviço tenha sido com esses dois protocolos, não haveria impedimento para expandir as alternativas com outros protocolos como MQTT, Profinet e etc.

De acordo com a Figura 51, o microserviço M2M possui basicamente duas ações. As ações estão relacionadas às funcionalidades principais do serviço: Aquisição de uma entrada (canal de I/O) sob requisição via protocolo de comunicação e Atualização de uma saída (canal de I/O) via protocolo de comunicação. As ações são replicadas para cada um dos protocolos suportados (Modbus TCP/IP e CoAP). Da mesma forma que no serviço DAQ, 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 51 – Microserviço M2M da MOA

Fonte: Autor

Microserviço CLP

Este microserviço é responsável por integrar as funcionalidades de um controlador lógico programável (CLP) na arquitetura do *Moleculer*. Na verdade, a proposta é adicionar um SoftPLC como serviço na estrutura do *Moleculer*. Um SoftPLC é um recurso que une as funcionalidades de um CLP com os benefícios de uma arquitetura aberta de *hardware* e *software*. Este serviço permitirá a execução de programas (lógicas e sequenciamento de tarefas) de CLPs, bem com o monitoramento de parâmetros e de I/Os do controlador e sendo proposto para ser hospedado ou rodar num sistema embarcado dedicado alocado no processo.

Esse serviço será desenvolvido com a integração do projeto OpenPLC (OPENPLC, 2017) à proposta MOA. O projeto OpenPLC segue as premissas da norma IEC 611311 e da PLCOpen XML2, 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 (*STL - Structured Text*), Lista de Instruções (*IL - Instruction List*), Sequenciamento Gráfico de Funções (*SFC - Sequential Function Chart*). A programação do *hardware*/CLP em

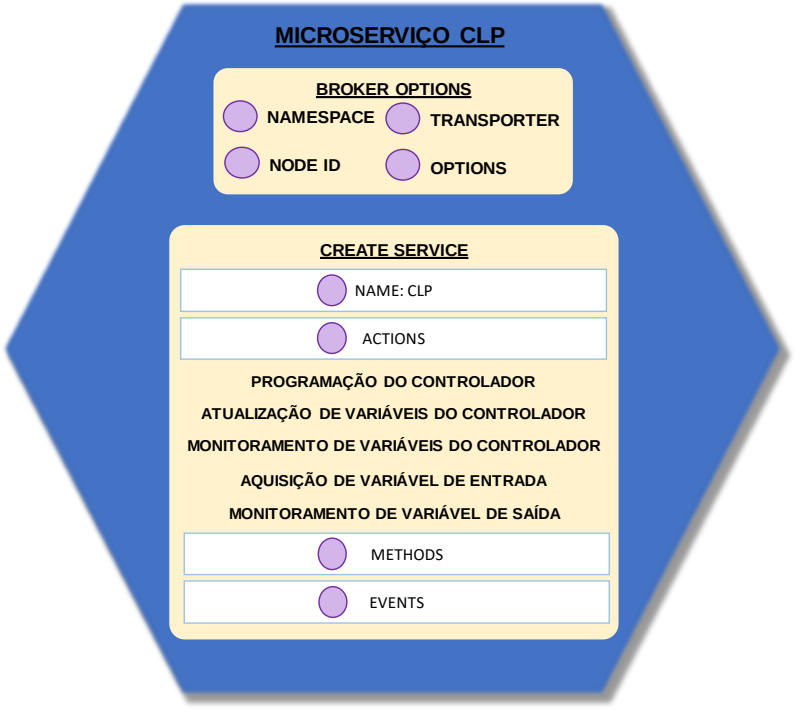
[1] IEC 61131 é um padrão internacional para controladores programáveis industriais, contemplando o projeto completo desde hardware, instalação, testes, documentação, programação e comunicação.

[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.

uma das linguagens do padrão IEC 61131-3 é feita no *PLCOpen Editor*, que é *software* de código aberto que faz parte do projeto *Beremiz* (BEREMIZ, 2017).

De acordo com a Figura 52, o microserviço CLP possui basicamente ações. As ações estão relacionadas às funcionalidades principais do serviço: Programação do controlador, 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 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 52 – Microserviço CLP da Arquitetura MOA



Fonte: Autor

5.1.1.2 Microserviços de Processos/Negócios

Os Microserviços de Processos/Negócios executam as funcionalidades de alto nível e geralmente necessitam operar em composição com outros serviços.

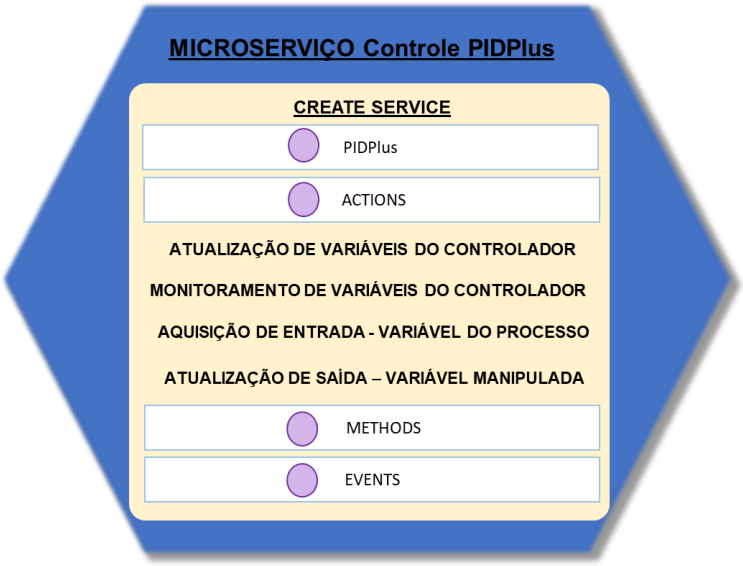
Microserviço Controle PIDPlus

O microserviço Controle PIDPlus tem como funcionalidade principal o controle de processos, sendo responsável por calcular, a partir de um algoritmo de controle, um sinal de controle a ser aplicado no processo a ser controlado. Cada instância desse serviço é responsável pela implementação de uma malha fechada de controle. Este serviço pode ser hospedado ou rodar em três tipos de plataforma: computador (PC), sistema embarcado ou nuvem. Este serviço necessita operar em conjunto com outros serviços (DAQ ou M2M) para obtenção das variáveis do processo a ser controlado.

O microserviço Controle PIDPlus utiliza como base o algoritmo de controle PIDPlus (seção 6.1.4) para cálculo das ações de controle. Este algoritmo foi criado para aplicações de controle em malha fechada usando redes de comunicação, mas neste trabalho foi modificado para trabalhar com o conceito de requisições e operar como um serviço. Apesar desse serviço usar especificamente o algoritmo de controle PIDPlus, um ponto importante 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.

De acordo com a Figura 53, o microserviço Controle PIDPlus possui basicamente ações. As 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 53 – Microserviço Controle PIDPlus da MOA



Fonte: Autor

5.1.1.3 Aplicações Internas e Externas

Na MOA da Figura 48, os serviços de Infraestrutura e de Processos/Negócios podem ser utilizados para criação de diferentes aplicações requeridas para operação do sistema. Na estrutura do *framework Moleculer*, a composição de serviços pode ser realizada usando o conceito de Orquestração (vide seção 2.6.3) ou de Coreografia (vide seção 2.6.4). Portanto, o *Moleculer* fornece uma estrutura padronizada que viabiliza e simplifica a composição dos microserviços desenvolvidos, de forma a facilitar a criação de aplicações mais complexas.

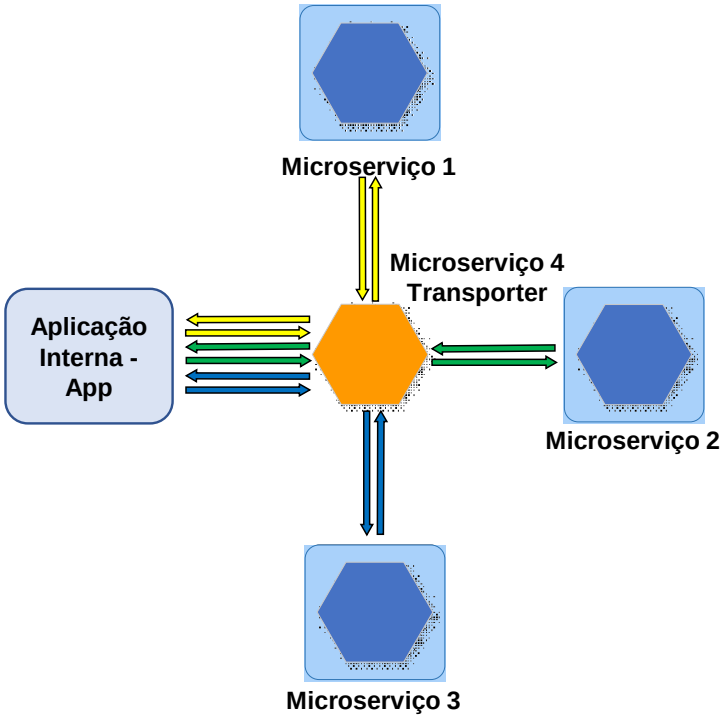
Na proposta deste trabalho, as aplicações na MOA podem ser criadas por meio de dois tipos de composição de microserviços, sendo chamadas de Aplicação Interna e Aplicação Externa. O *framework Moleculer* permite que essas aplicações sejam estruturadas de formas que os microserviços estejam alocados de duas formas: Distribuída (Figura 29) e Mista (Figura 30).

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). É importante verificar que a comunicação entre os microserviços via *Transporter* é automática e transparente, independendo do protocolo escolhido. Isso facilita o desenvolvimento das aplicações, pois não é necessário se preocupar com desenvolvimento de código pelo envio e recepção de mensagens em rede. 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.

As Aplicações Internas correspondem às aplicações que são integralmente desenvolvidas nas plataformas suportadas pelo *Moleculer* (JavaScript - Node.js, Java, .NET e Go – MOLECULER, 2018b). 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 54 apresenta um exemplo de orquestração de microserviços de uma Aplicação Interna na MOA usando o *Moleculer*. É possível verificar que esta Aplicação Interna utiliza 4 microserviços e que a Orquestração define a execução dos microserviços na sequência {App,4,1,4,App,4,2,4,App,4,3,4,App}. 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 (T*ransporter)* para estabelecer a comunicação entre si.

Figura 54 – Exemplo de uma Aplicação Interna por Orquestração de Serviços na MOA: Cada Microserviço Alocado em um Nó



Fonte: Autor

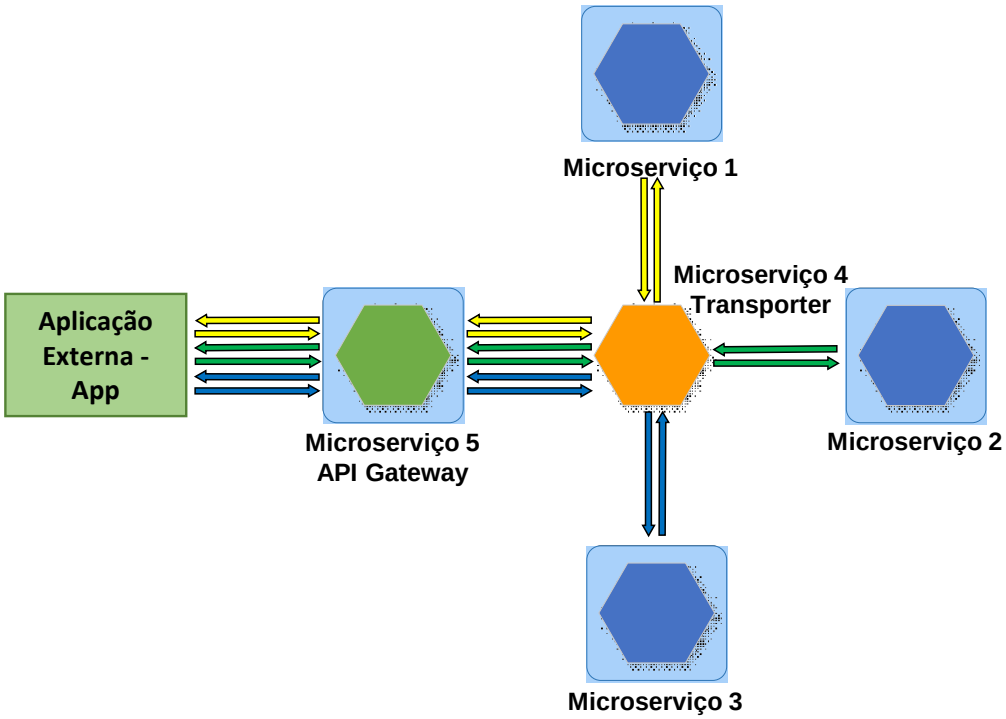
É possível verificar que na orquestração da Figura 54, a Aplicação Interna é o gerenciador (maestro) da execução dos outros microserviços. Cada serviço é requisitado, e após a resposta desse serviço, a Aplicação requisita a execução de um próximo serviço e assim sucessivamente. Portanto, é possível verificar que a sequência de execução de serviços definida na orquestração depende da alocação dos microserviços nos Nós do *Moleculer*.

As Aplicações Externas correspondem as aplicações que são desenvolvidas em qualquer plataforma de desenvolvimento diferente das suportadas pelo *Moleculer*. Neste tipo de aplicação, a orquestração dos microserviços ou sequência de operações definida sempre necessita ser realizada via um *software* (programa) externo usando comunicação REST via microserviço Gateway. Além disso, a comunicação entre os microserviços do *Moleculer* continua utilizando o microserviço *Transporter*.

De um modo geral, a estrutura utilizada para as Aplicações Externas são as mesmas utilizadas 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 55 apresenta um exemplo de orquestração de microserviços de uma Aplicação Externa na MOA usando o *Moleculer*. Esse exemplo é o mesmo da Figura 54, com a adição da execução da orquestração via um *software* externo.

Nessa configuração da Figura 55, 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.

Figura 55 - Exemplo de uma Aplicação Externa por Orquestração de Serviços na MOA: Cada Microserviço Alocado em um Nó



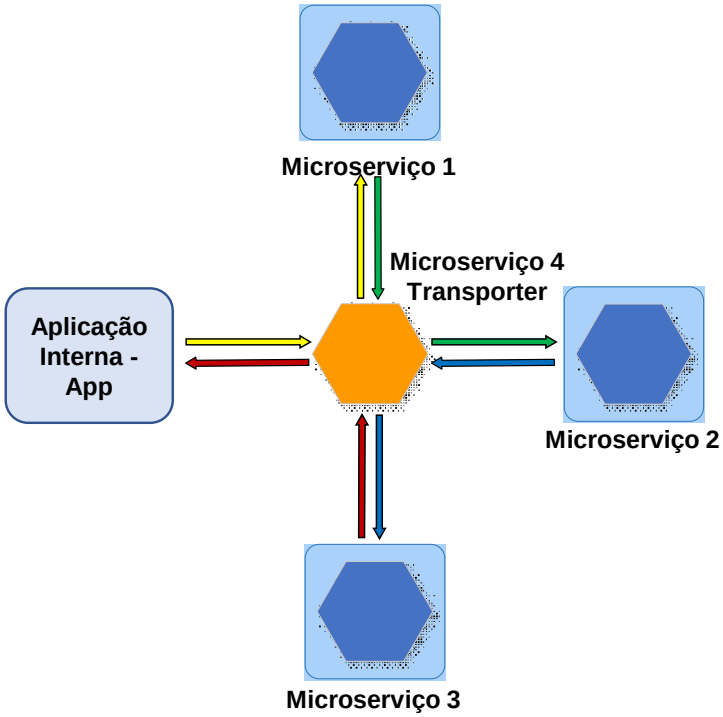
Fonte: Autor

A orquestração da Figura 55 define a execução dos microserviços na sequência {App,5,4,1,4,5,App,5,4,2,4,5,App,5,4,3,4,5,App}. 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 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.

No caso da composição de serviços por Coreografia, a diferença em relação à Orquestração é que os serviços são executados sequencialmente sem a necessidade da Aplicação, pois cada serviço sabe previamente qual o próximo serviço deve ser executado. A Figura 56 apresenta um exemplo de coreografia de microserviços de uma Aplicação Interna na MOA usando o *Moleculer*.

É possível verificar que esta Aplicação Interna utiliza 4 microserviços e que a Coreografia define a execução dos microserviços na sequência {App,4,1,4,2,4,3,4,App}. 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 (T*ransporter)* para estabelecer a comunicação entre si. É possível verificar que na Coreografia, a Aplicação Interna somente inicia a execução dos outros microserviços. Cada serviço requisitado é responsável por requisitar a execução do próximo serviço e assim sucessivamente até o final da composição.

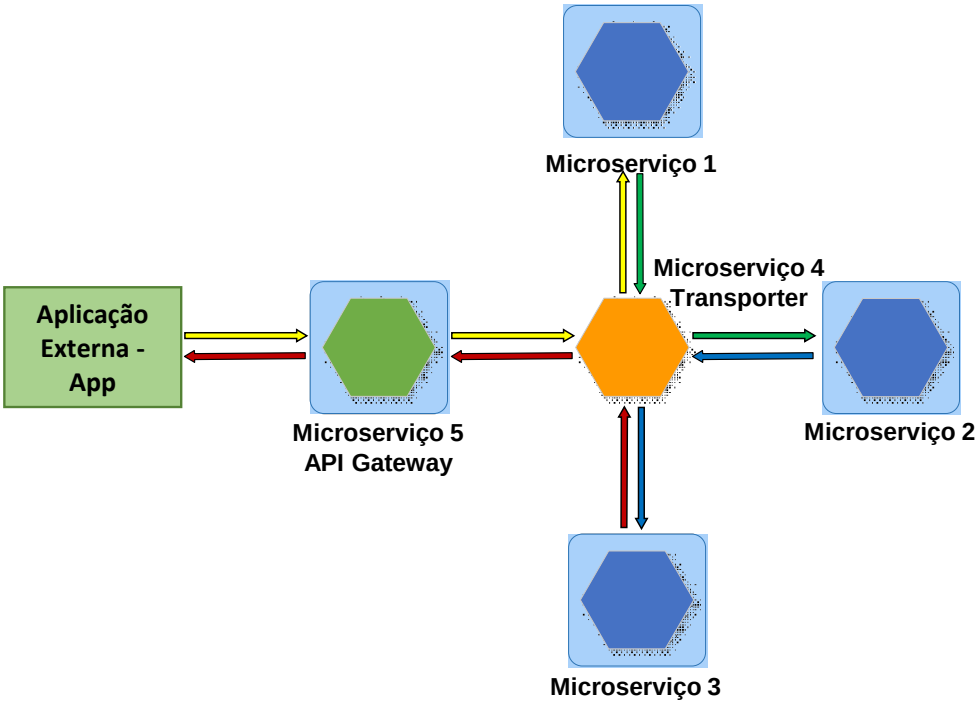
Figura 56 – Exemplo de uma Aplicação Interna por Coreografia de Serviços na MOA: Cada Microserviço Alocado em um Nó



Fonte: Autor

A Figura 57 apresenta um exemplo de coreografia de microserviços de uma Aplicação Externa na MOA usando o *Moleculer*. Esse exemplo é o mesmo da Figura 56, com a adição da execução da coreografia via um *software* externo. Nessa configuração da Figura 57, 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 {App,5,4,1,4,2,4,3,4,5,App}. 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 entre si.

Figura 57 - Exemplo de uma Aplicação Externa por Coreografia de Serviços na MOA: Cada Microserviço Alocado em um Nó



Fonte: Autor

A compreensão da influência da localização dos microserviços na execução da coreografia da aplicação criada é essencial para o correto entendimento da operação da MOA proposta e dos casos de estudo de aplicação da arquitetura que serão apesentados.

5.2 SÍNTESE DO CAPÍTULO

Este capitulo apresentou uma nova proposta de arquitetura orientada a Microserviços (MOA) para aplicações de IIoT e I4.0. Foi descrito a estrutura geral da MOA que utiliza para o seu desenvolvimento o *framework Moleculer* e discutido o desenvolvimento de diferentes microserviços para realização de atividades neste contexto de aplicação. Um ponto importante discutido neste capitulo é que esta MOA pode ser facilmente expandida, através da criação de novos microserviços e composição por coreografia dos serviços para permitir a realização de novas funcionalidades.

6 DESENVOLVIMENTO DA ARQUITETURA

Este capítulo descreve o desenvolvimento dos microserviços propostos nas plataformas utilizadas e apresenta as aplicações criadas a partir da orquestração dos serviços.

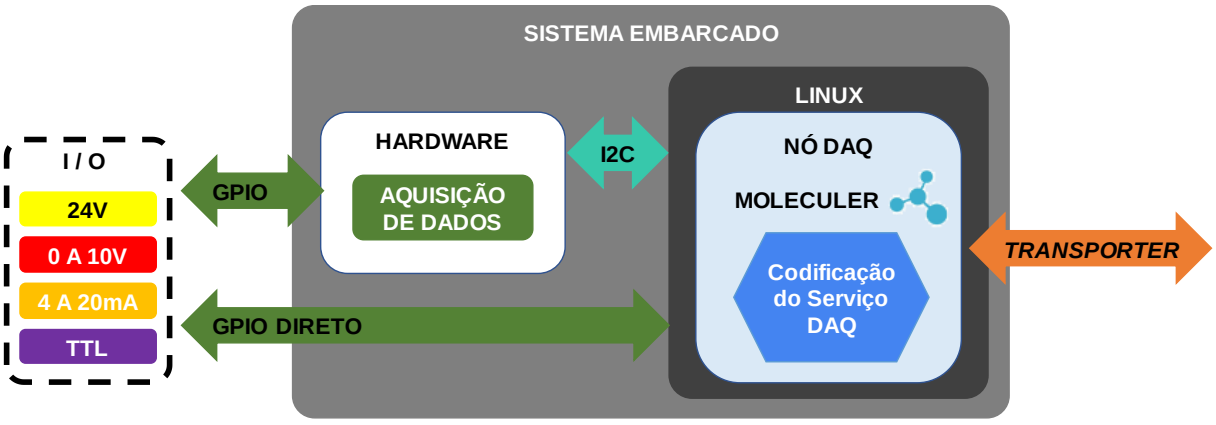
6.1 IMPLEMENTAÇÃO DOS SERVIÇOS E COMPOSIÇÃO DE APLICAÇÕES

Esta seção apresenta detalhes da implementação de alguns microserviços e aplicações propostos na seção 5.1.1 e 5.1.1.2. Estas implementações são posteriormente utilizadas nos casos de estudo apresentados no capítulo 7.

6.1.1 Microserviço DAQ

O microserviço DAQ desenvolvido neste trabalho possui a estrutura mostrada na Figura 58. A plataforma usada para operação do Nó DAQ foi um sistema embarcado composto por uma Raspberry Pi 3 rodando Linux (*Raspbian* – uma distribuição de Linux específica para *hardware* Raspberry Pi). O Nó DAQ desenvolvido é composto pelo *software* do *framework Moleculer* e da codificação do Serviço DAQ, que implementa as funcionalidades e ações do serviço (descritas na seção 5.1.1.1).

Figura 58 – Estrutura de Implementação do Nó e Serviço DAQ



Fonte: Autor.

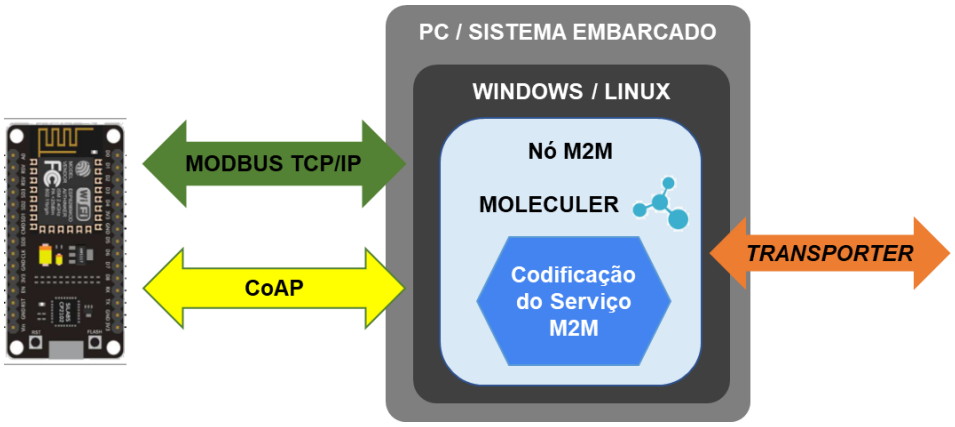
O sistema embarcado possui um *hardware* para conexão de entradas e saídas digitais e analógicas, responsável pela aquisição dos sinais elétricos correspondes às variáveis de interesse. Esses sinais são recebidos pelo *hardware* principal, que disponibiliza estas informações aos outros microserviços.

Os dados coletados são armazenados no Nó DAQ e a comunicação desses dados entre os microserviços é realizada pelo serviço de mensagens (*Transporter*), o qual deve estar hospedado/rodando em qualquer outro computador ou sistema embarcado na mesma infraestrutura de comunicação.

6.1.2 Microserviço M2M

O microserviço M2M desenvolvido neste trabalho possui a estrutura mostrada na Figura 59. O Nó M2M pode operar tanto num computador rodando Windows/Linux quanto num sistema embarcado composto por uma Raspberry Pi 3 rodando Linux. O único requisito da plataforma é a capacidade de comunicação Ethernet TCP/IP ou WiFi. O Nó M2M desenvolvido é composto pelo *software* do *framework Moleculer* e da codificação do Serviço M2M, que implementa as funcionalidades e ações do serviço (descritas na seção 5.1.1.1).

Figura 59 – Estrutura de Implementação do Nó e Microserviço M2M



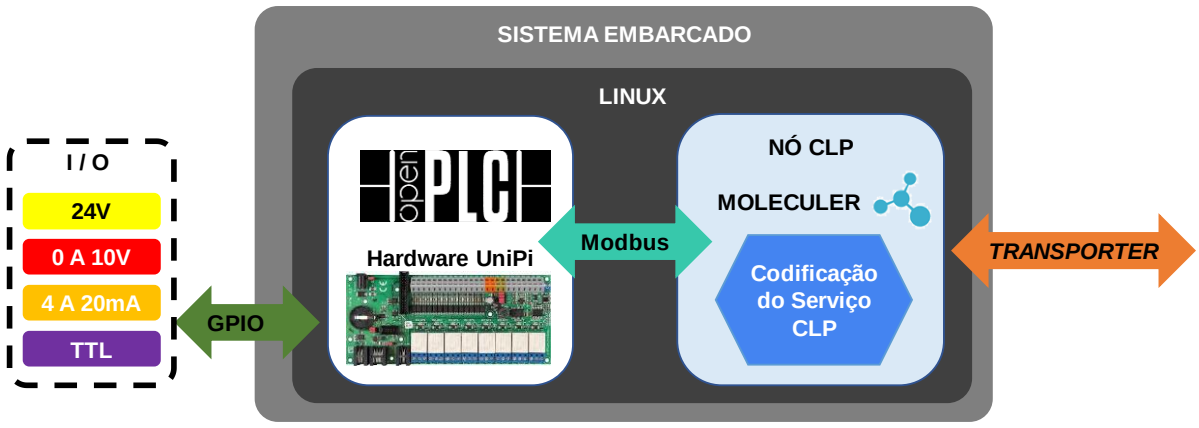
Fonte: Autor.

O Nó M2M se comunica com um *hardware* externo via protocolos de comunicação Modbus TCP/IP e CoAP. O *hardware* externo é responsável pela aquisição de entradas e saídas digitais e analógicas, e comunicação com o Nó M2M, que disponibiliza estas informações aos outros microserviços. Considerando que o *Moleculer* roda em plataforma Node.js, para o microserviço M2M foram utilizados os módulos *jsModbus* (JSMODBUS, 2018) para comunicação com módulos remotos por meio do protocolo Modbus TCP/IP e node-coap (NODE-COAP, 2018) para comunicação utilizando o protocolo CoAP.

6.1.3 Microserviço CLP

O microserviço CLP desenvolvido neste trabalho possui a estrutura mostrada na Figura 60. O Nó CLP foi desenvolvido para operar num sistema embarcado composto por uma Raspberry Pi 3 rodando Linux. O Nó CLP desenvolvido é composto pelo software do *framework Moleculer* e da codificação do Serviço CLP, que implementa as funcionalidades e ações do serviço (descritas na seção 5.1.1.1). Este microserviço fornece as funcionalidades de um Controlador Lógico Programável (CLP) com a vantagem de estar disponibilizado com um serviço. Para o desenvolvimento desse microserviço foi realizada a integração do projeto OpenPLC (OPEN PLC, 2019) junto à arquitetura de microserviço, conforme apresentado na Figura 60.

Figura 60 – Estrutura de Implementação do Nó e Microserviço CLP



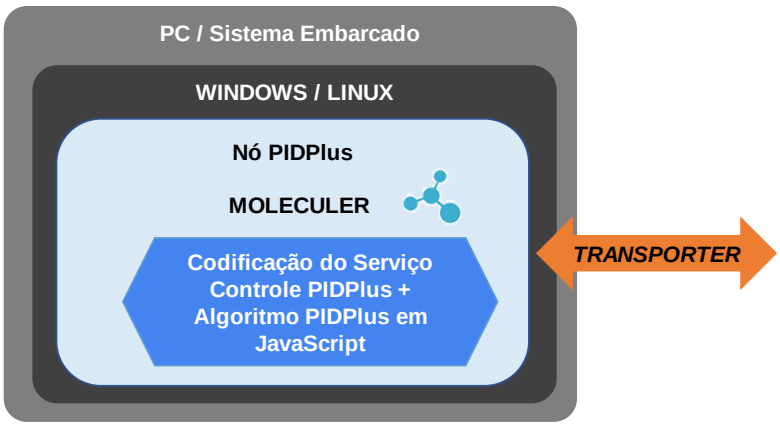
Fonte: Autor.

Através do *OpenPLC Editor*, que é uma IDE de desenvolvimento e programação de CLP compatível com a norma IEC 61131-3, o usuário do microserviço CLP pode programar a sua lógica de controle usando uma das 5 linguagens padronizadas. Este programa desenvolvido é compilado e carregado remotamente no servidor do OpenPLC para execução. O hardware usado neste projeto para interface e aquisição de entradas e saídas digitais e analógicas do processo a ser controlado foi o UniPi 1.1, que representa uma placa de expansão de IO compatível com Raspberry Pi. O OpenPLC é responsável pela execução da lógica de controle e atualização das entradas e saídas do hardware e disponibiliza num mapa de memória Modbus os valores/estados de cada uma dessas entradas e saídas do hardware UniPi 1.1 (UNIPI, 2019). Dessa forma, o código do serviço CLP implementado fica responsável por monitorar e atualizar as variáveis de operação do CLP (OpenPLC), usando comunicação Modbus, e disponibilizá-los de forma compatível para acesso via *framework Moleculer*.

6.1.4 Microserviço Controle PIDPlus

O microserviço Controle PIDPlus desenvolvido neste trabalho possui a estrutura mostrada na Figura 61 –O Nó PIDPlus pode operar tanto num computador rodando Windows/Linux quanto num sistema embarcado composto por uma Raspberry Pi 3 rodando Linux. O único requisito da plataforma é a capacidade de comunicação Ethernet TCP/IP ou WiFi. O Nó PIDPlus desenvolvido é composto pelo software do *framework Moleculer* e da codificação do Serviço Controle PIDPlus, que implementa as funcionalidades e ações do serviço (descritas na seção 5.1.1.2). É importante verificar que o algoritmo de controle PIDPlus foi modificado para funcionar via requisições e implementado em *JavaScript* direto na estrutura do *Moleculer* para melhorar o desempenho de execução desse serviço.

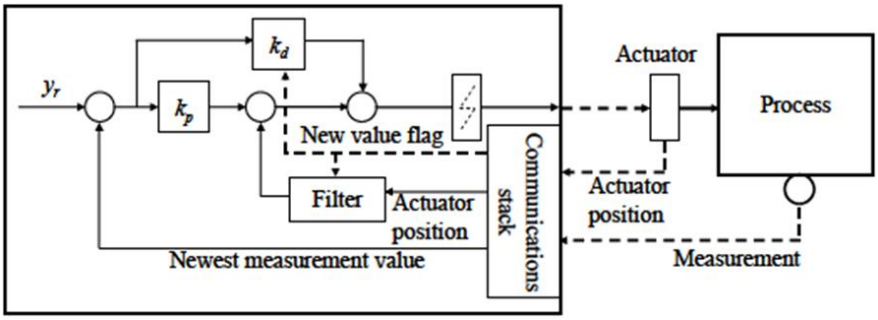
Figura 61 – Estrutura de Implementação do Nó e Serviço Controle PIDPlus



Fonte: Autor.

A teoria do algoritmo tradicional PIDPlus (SONG et al., 2006) é ilustrada na Figura 62. O PIDPlus mantém o sinal de controle no último nível calculado até que uma nova medida seja recebida. É importante notar que sua sintonia é independente do período de amostragem, dependendo apenas das características físicas da planta.

Figura 62 - Estrutura do controlador PIDPlus.



Fonte: Song et al. (2006)

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 foi substituído por um filtro de 1ªordem. A saída do filtro é calculada conforme (1):

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

Onde: $\mathbf{F_N}$ = nova saída do filtro, $\mathbf{F_{(N-1)}}$ = saída do filtro na última execução, $\mathbf{O_{(N-1)}}$ = saída do controlador na última execução e $\Delta\mathbf{T}$ = intervalo de tempo desde que o último valor medido foi recebido e $\mathbf{T_{reset}}$ = constante de tempo da planta somado ao tempo morto. O tempo integrativo (**Ti**) do controlador também pode ser usado como um parâmetro RESET simplificado. A parte derivativa é substituída conforme (2):

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

Onde: $\mathbf{e_N}$= erro atual, $\mathbf{e_{(N-1)}}$ = último erro, $\mathbf{O_D}$ = termo derivativo do controlador e $\mathbf{K_D}$ = ganho derivativo.

Considere a contribuição da parte derivativa quando as entradas são perdidas durante vários períodos. Para o algoritmo PID tradicional, o divisor na parte derivativa seria o período (discretização do controlador), enquanto que, no algoritmo PIDPlus é o tempo decorrido entre duas medições recebidas com sucesso ($\Delta\mathbf{T}$). É óbvio que o algoritmo modificado produz uma ação derivativa menor do que o algoritmo de controle PID (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.

6.2 IMPLEMENTAÇÃO DAS APLICAÇÕES

Esta seção apresenta detalhes da implementação das aplicações propostos na seção 5.1.1.3. Estas implementações são posteriormente utilizadas nos casos de estudo apresentados no capítulo 7.

Apesar de a arquitetura do *Moleculer* permitir a criação das Aplicações Internas (desenvolvidas em Node.js) e Externas (via API Gateway) através da composição de serviços por Orquestração e por Coreografia, neste trabalho foi definido somente o uso da Orquestração para criação das aplicações. Em se tratando da alocação dos serviços, o uso da arquitetura distribuída onde cada serviço é alocado em um nó diferente foi utilizada.

Além disso, como era necessária uma interface gráfica para visualização do resultado do processo controlado (variáveis no tempo) e interação com o sistema (mudança de parâmetros), para facilitar o desenvolvimento optou-se por somente desenvolver Aplicações Externas. Para uma Aplicação Interna, essa interface gráfica de visualização e interação teria que ser desenvolvido em *JavaScript* ambiente Node.js. Para este trabalho, o software LabVIEW foi utilizado para desenvolvimentos das Aplicações Externas.

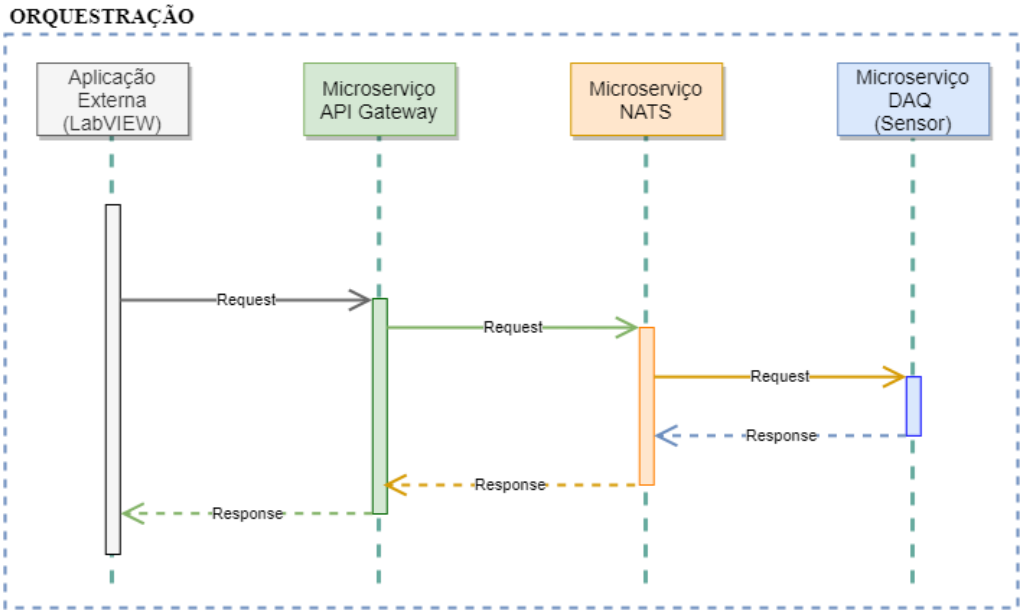
6.2.1 Aplicação Externa de Supervisão e Monitoramento

De forma a viabilizar o monitoramento e visualização de dados de processo usando a arquitetura MOA, uma Aplicação Supervisão e Monitoramento foi criada a partir da orquestração dos microserviços. A ideia dessa aplicação é a criação de uma interface para visualização dos dados das variáveis do processo e a análise do comportamento das variáveis ao longo do tempo. A sequência de execução dos microserviços dessa Aplicação Supervisão e Monitoramento depende da localização de alocação dos serviços nos Nós do *Moleculer*. Além disso, a Aplicação pode utilizar para amostragem e atualização das variáveis do processo tanto o microserviço DAQ quanto o M2M.

Aplicação de Monitoramento e Supervisão usando Serviço DAQ

A Figura 63 apresenta a sequência de orquestração de serviços da Aplicação Externa de Supervisão e Monitoramento usando o serviço DAQ para amostragem e atualização das variáveis do processo. Nessa Aplicação Externa, a orquestração de serviços é executada externamente à plataforma do *Moleculer*, usando o LabVIEW. Consequentemente, nesse caso a comunicação e integração dos serviços é ser realizada via serviço API Gateway usando comunicação via REST.

Figura 63 - Orquestração de Serviços: Aplicação Monitoramento e Supervisão usando Serviço DAQ



Fonte: Autor

Considerando a Aplicação Externa de Supervisão e Monitoramento da Figura 63, o microserviço DAQ utiliza a ação de Aquisição de entrada para envio dos dados das variáveis desejadas. A criação de um dashboard para visualização do histórico de variação de variáveis de interesse no tempo seria facilitada pelo uso dessa aplicação de monitoramento.

No caso da uso dessa aplicação de monitoramento e supervisão para uma malha de controle de processo, além da orquestração mostrada na Figura 63 que forneceria os dados da variável do processo, seria necessário também a orquestração do serviço PIDPlus para requisição das variáveis de interesse da malha. Nesse caso, adicionalmente o microserviço Controle PIDPlus utiliza a ação de Monitoramento de Variáveis do Controlador (Setpoint, Variável do processo e Variável manipulada) para envio das variáveis requeridas da malha de controle.

Aplicação de Monitoramento e Supervisão usando Serviço M2M

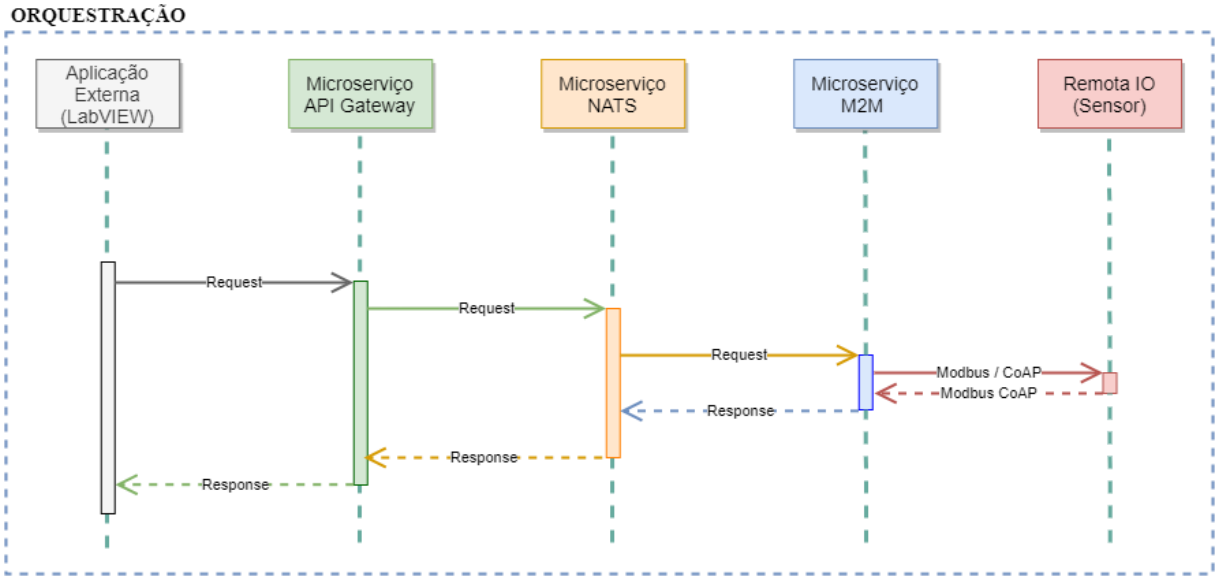
A Figura 64 apresenta a sequência de orquestração de serviços para uma Aplicação Externa de Supervisão e Monitoramento usando o serviço M2M para amostragem e atualização das variáveis do processo. Percebe-se que a única diferença desta orquestração para a da Figura 63 é a alteração do microserviço DAQ pelo M2M. O microserviço M2M utiliza a ação de Aquisição de entrada sob requisição via protocolo para envio dos dados das variáveis desejadas.

É importante verificar que o serviço M2M permite a comunicação via Modbus TCP/IP e via CoAP com hardware externo. Dessa forma, conforme a escolha do protocolo, as ações do serviço M2M devem ser selecionadas em relação a esse protocolo. Esta possibilidade fornece

versatilidade para uso da MOA e do serviço M2M com qualquer tipo de remota de IO compatível com comunicação nesses dois protocolos.

Da mesma forma, no caso de uso dessa aplicação de monitoramento e supervisão para uma malha de controle de processo, além da orquestração mostrada na Figura 64 que forneceria os dados da variável do processo, seria necessário também a orquestração do serviço PIDPlus para requisição das variáveis de interesse da malha. Nesse caso, adicionalmente o microserviço Controle PIDPlus utiliza a ação de Monitoramento de Variáveis do Controlador (Setpoint, Variável do processo e Variável manipulada) para envio das variáveis requeridas da malha de controle.

Figura 64 - Orquestração de Serviços: Aplicação Monitoramento e Supervisão usando Serviço M2M

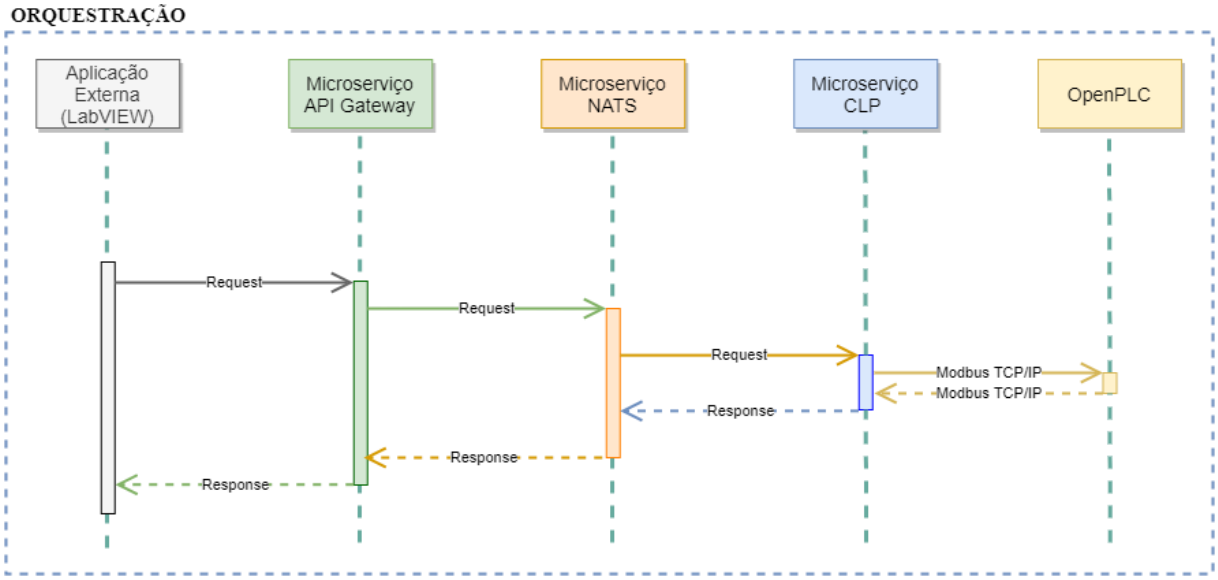


Fonte: Autor

Aplicação de Monitoramento e Supervisão usando Serviço CLP

A Figura 65 apresenta a sequência de orquestração de serviços para uma Aplicação Externa de Supervisão e Monitoramento usando o serviço CLP para amostragem e atualização das variáveis do processo e do controlador CLP. É importante verificar que nesta aplicação o controle é implementado localmente através do OpenPLC, portanto também há a possibilidade do desenvolvimento de uma lógica de controle para o sistema em questão. Percebe-se que a única diferença desta orquestração para a da Figura 64 é a alteração do microserviço M2M pelo CLP. Lembrando que o microserviço CLP obtém os valores das variáveis do controlador CLP através de uma comunicação Modbus TCP/IP (seção 6.1.3). O microserviço CLP utiliza as ações de Programação do controlador, Atualização de variáveis do controlador e Monitoramento de variáveis do controlador.

Figura 65 - Orquestração de Serviços: Aplicação Monitoramento e Supervisão usando Serviço CLP



Fonte: Autor

6.2.2 Aplicação Externa Controle de Processo

De forma a viabilizar a implementação de malhas de automação e controle usando a MOA, uma Aplicação Controle de Processo foi criada a partir da orquestração de microserviços. A sequência de execução dos microserviços dessa Aplicação Controle de Processo depende da localização de alocação dos serviços nos Nós do *Moleculer*. Além disso, a Aplicação pode utilizar para amostragem e atualização das variáveis do processo tanto o microserviço DAQ quanto o M2M.

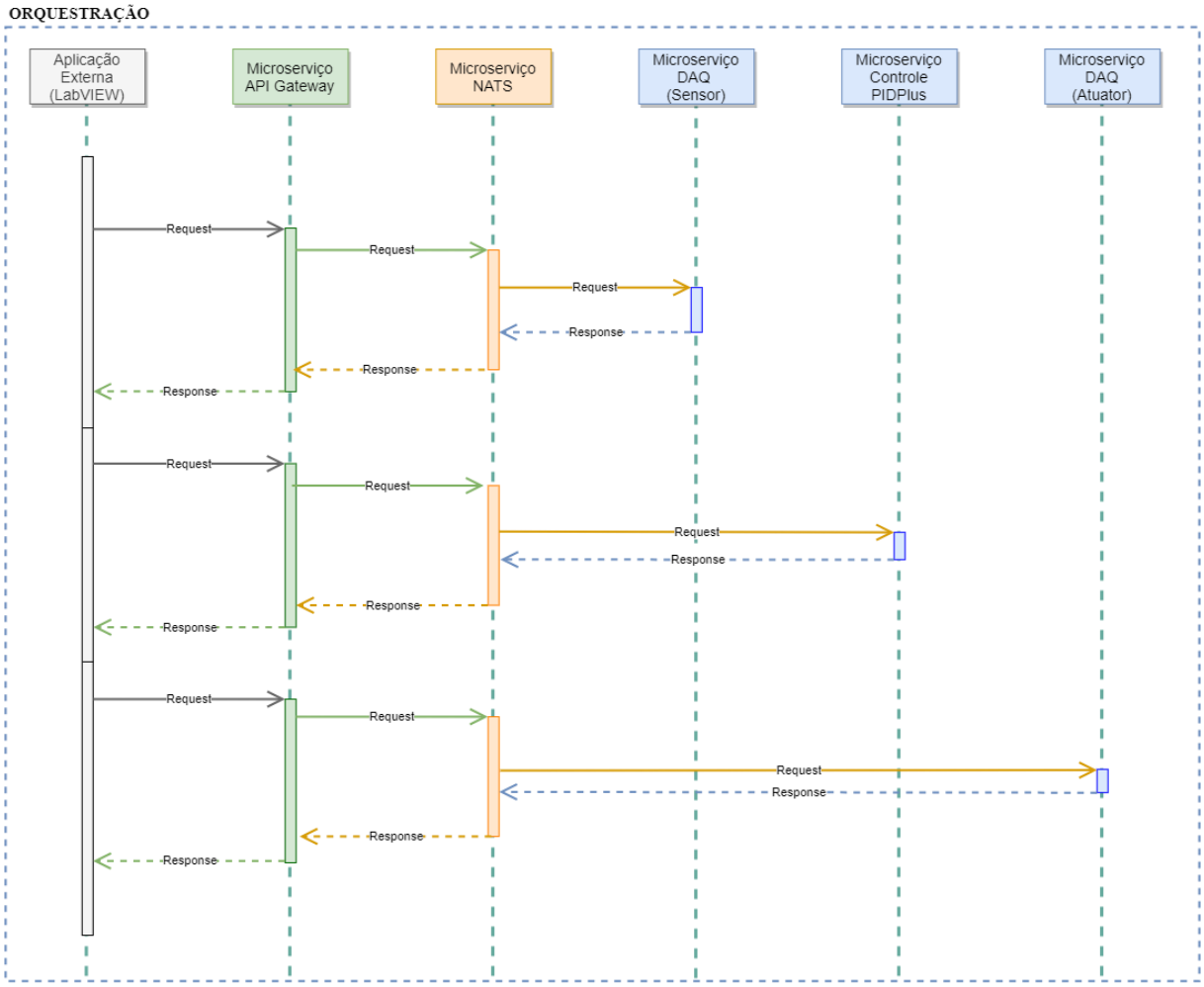
Aplicação Controle de Processo usando Serviço DAQ

A Figura 66 apresenta a sequência de orquestração de serviços a Aplicação Externa de Controle de Processo usando o serviço DAQ para amostragem e atualização das variáveis do processo. Nessa Aplicação Externa, a orquestração de serviços é executada externamente à plataforma do *Moleculer*. Consequentemente, nesse caso a comunicação e integração dos serviços é realizada via serviço API Gateway usando comunicação via API REST.

O microserviço DAQ utiliza duas ações: Aquisição de Entrada e Atualização de Saída. O microserviço Controle PIDPlus utiliza as ações para recebimento das variáveis do controlador e transmissão da ação de controle calculada. Na orquestração de serviços, na qual verifica-se que cada microserviço está alocado num Nó diferente, a cada tempo de ciclo ou atualização da malha de controle, a Aplicação Externa manda uma requisição (*request*) para o microserviço DAQ. O microserviço DAQ realiza a aquisição de dados de um canal de entrada e transmite esse dado de volta (*response*) para a Aplicação Externa. A Aplicação Externa envia esse dado

de entrada da variável de interesse, juntamente com os parâmetros de configuração do controlador, para o microserviço Controle PIDPlus através de uma nova requisição. O microserviço Controle PIDPlus executa o algoritmo de controle PIDPlus, e calcula um novo valor do sinal de controle para ser aplicado ao atuador da malha. O sinal de controle calculado é então transmitido de volta para a Aplicação Externa através de uma resposta. Finalmente, a Aplicação envia esse dado do sinal de controle para o microserviço DAQ, que executa a ação de atualizar uma saída ou canal de saída do hardware para atuação sobre a planta. Conforme informado, todo esse processo de orquestração é repetido ciclicamente de acordo com o tempo de ciclo definido para a malha de controle.

Figura 66 - Orquestração de Serviços: Aplicação Externa Controle de Processo usando Serviço DAQ



Fonte: Autor

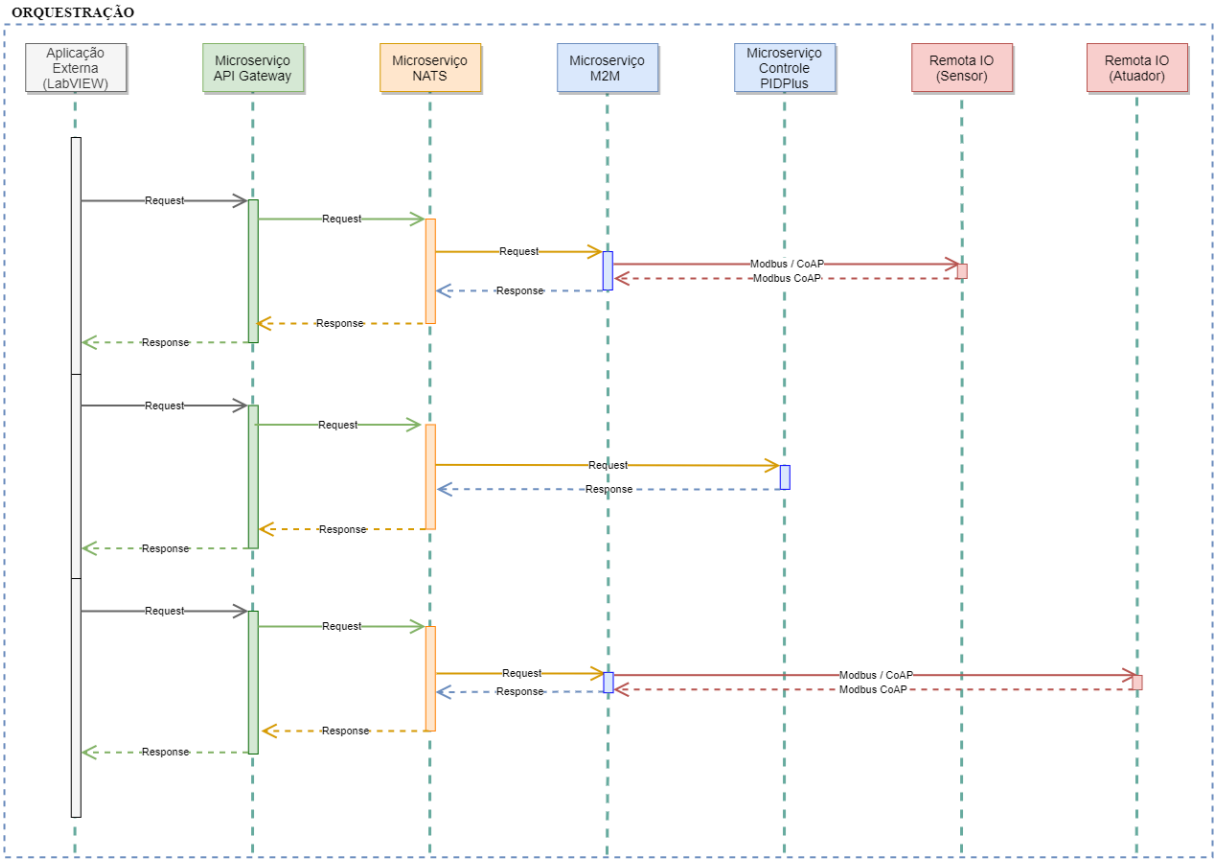
Aplicação Controle de Processo usando Serviço M2M

A Figura 67 apresenta a sequência de orquestração de serviços para a Aplicação Externa de Controle de Processo usando o serviço M2M para amostragem e atualização das variáveis

do processo. Percebe-se que a única diferença desta orquestração para a da Figura 66 é a alteração do microserviço DAQ pelo M2M.

Da mesma forma, a explicação e descrição da sequência de execução dos serviços é mesma e não precisa ser novamente apresentada. É importante verificar que o serviço M2M permite a comunicação via Modbus TCP/IP e via CoAP com hardware externo. Dessa forma, conforme a escolha do protocolo, as ações do serviço M2M devem ser selecionadas em relação a esse protocolo. Esta possibilidade fornece versatilidade para uso da MOA e do serviço M2M com qualquer tipo de remota de IO compatível com comunicação nesses dois protocolos.

Figura 67 - Orquestração de Serviços: Aplicação Externa Controle de Processo usando Serviço M2M



Fonte: Autor

Ainda que neste trabalho o desenvolvimento de uma Aplicação Interna não tenha sido utilizado, a única diferença em comparação com as Aplicações Externas mostradas anteriormente, seria que a Aplicação Interna comunicaria diretamente via serviço *Transporter* (NATS), não sendo necessário o uso do serviço API gateway. Dessa forma, os diagramas das orquestrações apresentados seriam simplificados, apresentando menor quantidade de comunicação de dados entre serviços.

A análise das aplicações desenvolvidas permite constatar a grande vantagem proporcionada pela composição de serviços na MOA para aplicações de automação e controle para a Indústria 4.0. As inúmeras possibilidades de composição de serviços fornecem enorme flexibilidade para a criação das aplicações, diante da possibilidade de reuso dos serviços e criação de novos serviços e composições. Isso, atrelado ao fato da comunicação se tornar automática e transparente entre os microserviços, proporciona interoperabilidade total e supera problemas de integração de dados entre diferentes níveis hierárquicos das aplicações industriais. Alguns exemplos de aplicações desenvolvidas serão testados experimentalmente no próximo capitulo e avaliadas em relação ao seu potencial de aplicabilidade em sistemas de automação e controle industriais.

7 RESULTADOS E DISCUSSÕES

Este capítulo descreve os diversos experimentos realizados para avaliar a MOA, os microserviços e aplicações criadas a partir da orquestração dos serviços. Os experimentos foram realizados com a operação da MOA em diferentes configurações aplicadas ao controle de velocidade de um motor CC de uma planta didática real e de um modelo dessa planta didática simulada em tempo real usando a tecnologia *Hardware-In-the-Loop* (HIL).

7.1 DESCRIÇÃO DOS EXPERIMENTOS E APARATO EXPERIMENTAL

7.1.1 Planta Didática de Motor CC

Buscando a validação experimental da arquitetura MOA, uma planta didática de controle de velocidade de um motor CC foi utilizada. A planta a ser controlada é dada por um kit didático de controle de motor CC, composto por um motor *Motron* M910 de 24V e 43W, 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) e um circuito de acionamento do motor via sinal PWM. Esta planta didática foi usada em conjunto com os microserviços desenvolvidos neste trabalho para avaliar a operação da MOA.

Figura 68 – Planta Didática de Controle de Motor CC

Fonte: Autor.

7.1.2 Ambiente HIL

O estudo das características de sistemas dinâmicos através de simulações ganhou importância ao longo dos anos, e esta prática garante benefícios significativos para aplicações de automação e controle, como redução de custos com a prototipagem e a possibilidade de testar sistemas em diferentes condições com alta repetibilidade (SAHIN et al., 2012). Existem diferentes técnicas para realizar simulações de controle de processo. As três técnicas de simulação mais conhecidas são a Prototipagem de Rápida de Controle (RCP – *Rapid Control Prototyping*), o *Hardware-In-the-Loop* (HIL) e o *Software-In-the-Loop* (SIL) (ISERMANN, 2008). RCP e HIL são considerados simulações em tempo real.

HIL refere-se a um sistema onde alguns dos componentes reais são integrados com sistemas virtuais ou simulados. Considerando a técnica HIL, geralmente equipamentos reais (controlador, comunicação) estão conectados a uma planta virtual executada em um simulador de tempo real. A principal vantagem do HIL é permitir testes confiáveis em condições indisponíveis em plantas reais. Os modelos matemáticos podem substituir componentes reais e os componentes a serem testados são inseridos no circuito fechado.

A tecnologia de simulação HIL pode ser encontrada em quase todas as áreas onde testes mais realistas de componentes do sistema são necessários antes da sua construção final. As simulações de HIL são uma ferramenta muito útil para avaliar e desenvolver controladores, oferecendo um risco zero na experimentação de diferentes técnicas e metodologias de controle sem necessidade da planta real para testes. Desta forma, é possível economizar investimentos e evitar consequências perigosas, resultando em erros na fase de projeto.

O contexto da simulação HIL mostra vantagens de usar a emulação de modelos em tempo real em um ambiente dinâmico e integrado como o de aplicações de IIoT e I4.0. Dessa forma, este trabalho utilizou dessa possibilidade, criando um modelo simulado da planta didática de controle de motor CC, o qual pode ser replicado e utilizado nos experimentos, superando a limitação de disponibilidade de somente uma planta didática real. Além disso, o uso do HIL com a MOA neste trabalho proporciona grande flexibilidade e pode ser expandida para outros tipos de modelos de processos, algoritmos de controle e malhas de controle simultâneas, possibilitando a análise operacional e de desempenho da solução completa.

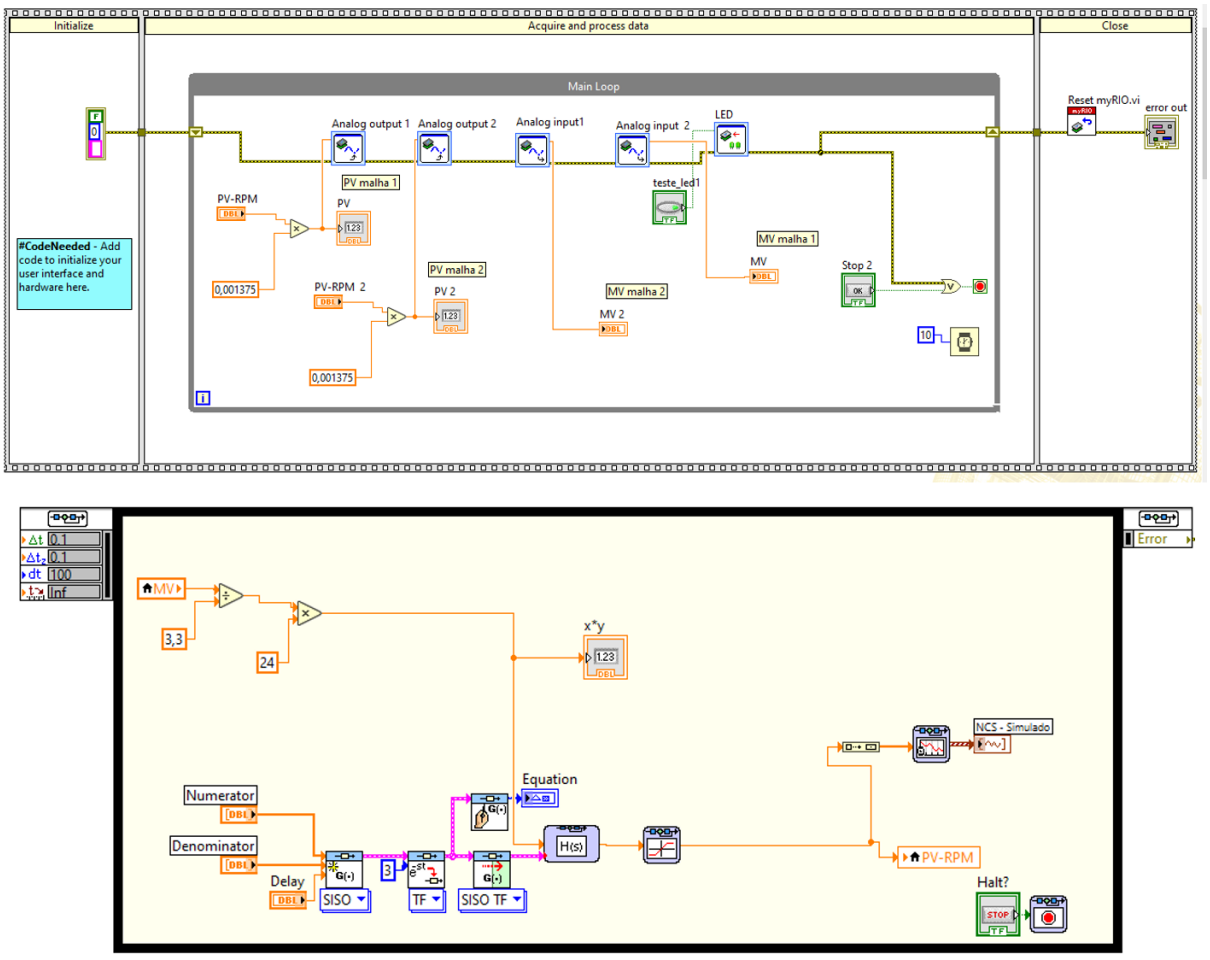
Na estrutura usada neste trabalho, a planta a ser controlada é representada por um modelo matemático do kit didático de controle de motor CC. O processo simulado é o motor CC do kit, o qual é executado no *LabVIEW Realtime* (RTOS – *Realtime Operational System*) usando a técnica HIL. O modelo de processo roda no MyRIO, que é um hardware dedicado da *National Instruments*, com processador de tempo real. O modelo matemático do motor CC foi

identificado experimentalmente e utilizado para as simulações HIL é dado pela função de transferência mostrada na Equação 1:

$$\frac{\omega_m}{V}(s) = \frac{141,5}{(1,5s+1)} e^{-0,35s} \quad (1)$$

Onde ω_m é a velocidade de rotação do motor e V é a tensão aplicada na armadura do motor CC. O modelo matemático da Equação 1 foi implementado com o *LabVIEW Control Design and Simulation*, de acordo com a Figura 69, e é executado em tempo real no hardware MyRIO.

Figura 69 – Implementação no LabVIEW do Modelo Matemático do Motor CC para Simulação HIL

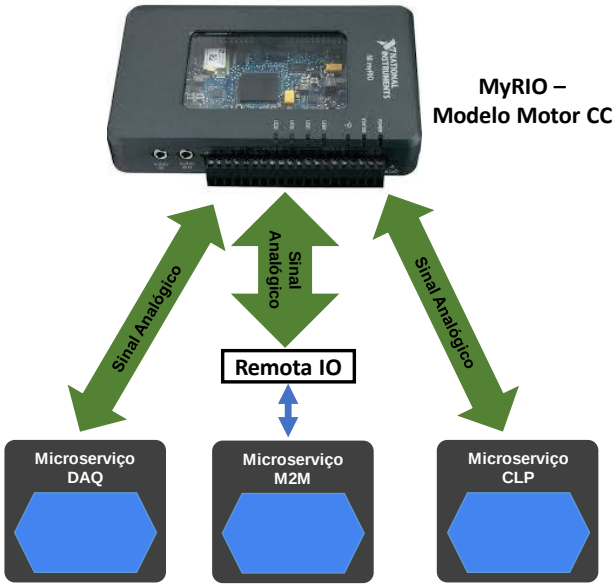


Fonte: Autor

No funcionamento do HIL, o dispositivo sensor da aplicação adquire (ADC) a variável de processo (PV) usando um sinal analógico de uma saída DAC do MyRIO. O dispositivo atuador também usa um sinal analógico para acionamento do motor (MyRIO). No caso da

MOA, o papel de sensor e atuador pode ser realizado pelos três microserviços de infraestrutura criados neste trabalho: microserviço DAQ, M2M e CLP, conforme mostrado na Figura 70.

Figura 70 - Conexão de Hardware da Simulação HIL com Microserviços



Fonte: Autor

7.1.3 Configurações dos Experimentos

Os experimentos que serão apresentados nas próximas subseções utilizaram a arquitetura MOA desenvolvida para controle da planta didática de motor CC ou do modelo simulado em HIL dessa planta. De forma a auxiliar nas análises dos resultados dos experimentos e permitir uma comparação entre eles, padronizaram-se as seguintes configurações e parâmetros de interesse para a realização dos experimentos:

- ✓ Utilização de comunicação sem fio usando rede Wi-Fi com dispositivo roteador para conexão entre todos os serviços e nós que compõe os experimentos;
- ✓ Utilização do mesmo perfil de setpoint (referência de velocidade), com variações positivas e negativas do tipo degrau e rampa e com o mesmo tempo de duração;
- ✓ Cálculo e estatísticas (média e desvio padrão) dos tempos de requisição e execução dos microserviços utilizados: trata-se do tempo entre o início da requisição do serviço pela aplicação, execução do serviço e recebimento da resposta do serviço pela aplicação;
- ✓ Cálculo e estatísticas (média e desvio padrão) dos tempos de ciclo de orquestração das malhas de controle implementadas: o tempo de ciclo da malha de controle corresponde ao período completo de execução de um laço da malha, na sequência sensor – controlador – atuador. Trata-se, portanto, do tempo total de orquestração da malha de controle,

considerando o tempo utilizado pela aplicação desde o início da requisição do serviço de aquisição de dados do sensor, execução do serviço de controle e recebimento da resposta do serviço de atualização do atuador.

É importante enfatizar que nesses experimentos realizados o objetivo não foi avaliar o desempenho de controle, mas sim 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, isso diante das diversas possibilidades de composição das aplicações usando os serviços desenvolvidos. Adicionalmente, objetivou-se a análise temporal (latência e jitter) e de desempenho da comunicação em rede da arquitetura MOA nos experimentos realizados.

7.2 EXPERIMENTO 1: CONTROLE USANDO SERVIÇOS DAQ E CONTROLE PIDPLUS

7.2.1 Orquestração de Serviços usando LabVIEW

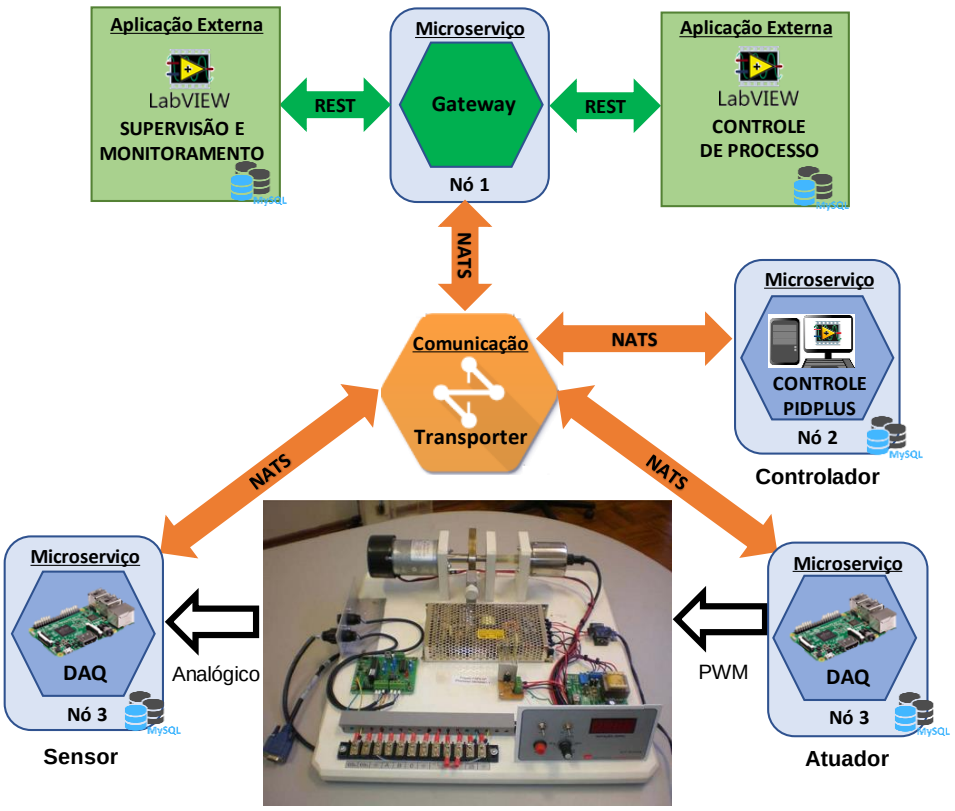
No primeiro experimento realizado, a malha de controle de velocidade de motor CC da planta didática foi utilizada com a arquitetura MOA. A estrutura geral desse experimento envolvendo a MOA é apresentada na Figura 71.

Na Figura 71, uma Aplicação Externa de “Controle de Processos” (seção 6.2.2) e uma de “Supervisão” (seção 6.2.1) foram criadas a partir da orquestração dos microserviços de Controle PIDPlus, DAQ, *Transporter*, Gateway e Supervisão. Essas aplicações foram desenvolvidas usando o software LabVIEW e são responsáveis pelo monitoramento (plotagem das curvas de resposta de saída), supervisão de variáveis e envio dos dados de controle do processo e por implementar o algoritmo de controle (PIDPlus) da malha. O serviço API *Gateway*, o serviço *Transporter* e as aplicações externas Controle e Supervisão são executados em um computador (PC). O banco de dados utilizado pelos serviços para armazenamento dos dados das variáveis é o MySQL.

O serviço DAQ (Nó 3) realiza a aquisição da variável de processo (PV) e atualização da variável manipulada (MV) na planta. O serviço DAQ é responsável pela aquisição de dados do sensor e do atuador. Um sistema embarcado (Raspberry Pi 3B+ com distribuição Raspbian Linux) foi usado para a implementação e operação do serviço DAQ. O sensor adquire a variável de processo (PV) usando um sinal analógico. O atuador é responsável por receber os sinais de controle da aplicação de Controle (Nó 2) e atuar na planta usando um canal PWM. O serviço de Controle PIDPlus (Nó 2) implementa o algoritmo PIDPlus em *JavaScript* para o cálculo do

controle de processo, a partir da variável do processo (dados do sensor - velocidade do motor) e atua no motor (dados do atuador - tensão aplicada ao motor).

Figura 71 - Arquitetura MOA para o Experimento de Controle de Processo da Planta Didática com Orquestração de Serviços via LabVIEW



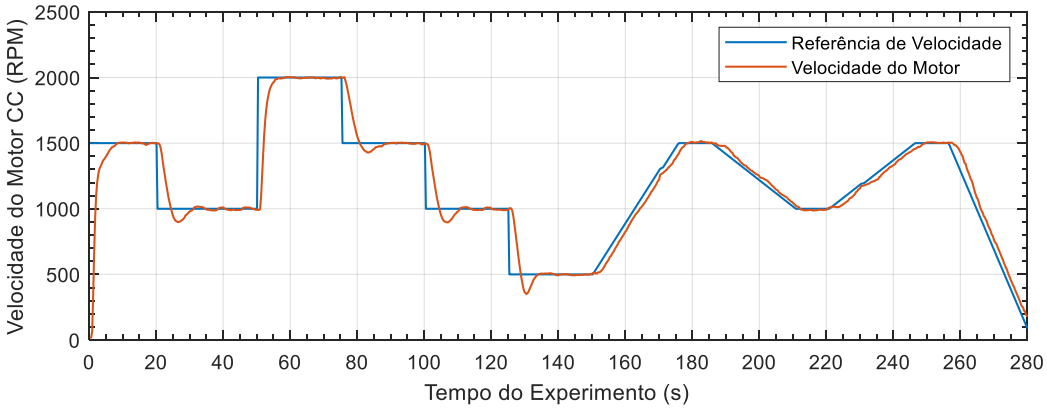
Fonte: Autor.

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 API *Gateway* (Nó 1) realiza a comunicação via REST entre a aplicação externa criada no LabVIEW com os microserviços desenvolvidos no *Moleculer*.

Testes operacionais com a MOA proposta na estrutura da Figura 71 foram realizados. O controle da malha fechada foi definido com um período de ciclo de 300ms (aquisição do sensor, computação do controle e atualização do sinal de controle na planta). A Figura 72 mostra os resultados da resposta do sistema de controle de velocidade do motor e a Figura 73 mostra o sinal de controle aplicado ao motor.

Ambas as respostas foram armazenadas pelo aplicativo Supervisão. As curvas de saída (Figura 72) mostram a resposta de velocidade do motor de acordo com a referência definida. A Figura 72 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.

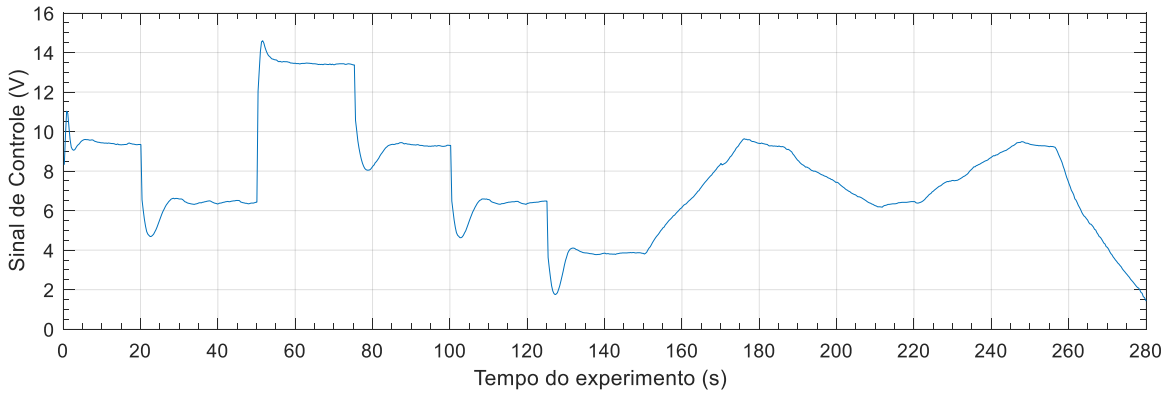
Figura 72 – Resposta do Experimento de Controle da Planta Didática com a Arquitetura MOA da Figura 71



Fonte: Autor.

A Figura 73 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 73 – Sinal de Controle do Experimento de Controle com a Arquitetura MOA da Figura 71.



Fonte: Autor.

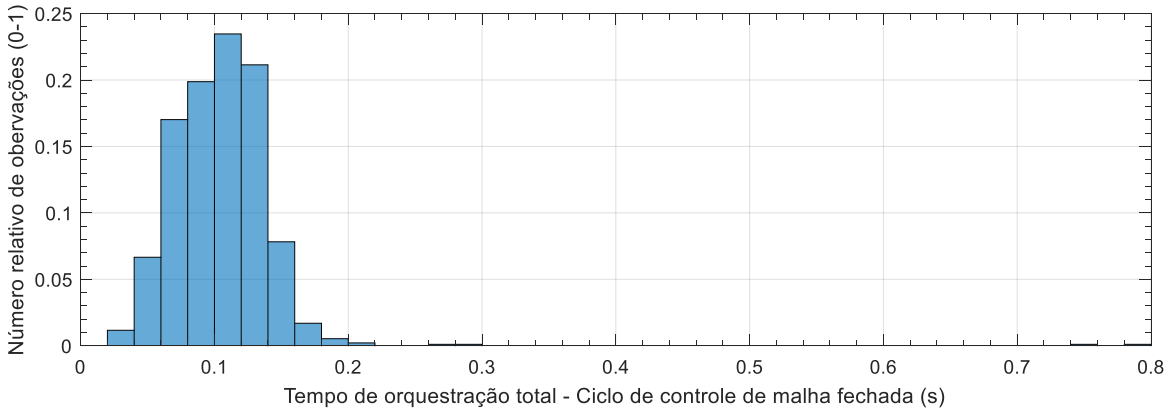
Existem várias métricas de desempenho de rede usadas para analisar sistemas de controle em rede. As principais métricas que afetam o desempenho do controle são o tempo médio de atraso e *jitter* (LIAN; MOYNE; TILBURY, 2002). O atraso é o tempo decorrido entre o envio de dados em um nó e o recebimento em outro. No caso do experimento de controle como um serviço apresentado na Figura 71, o tempo de atraso seria o tempo entre uma solicitação de rede do aplicativo externo e a execução do serviço.

É difícil medir experimentalmente o tempo de atraso 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 conforme explicado na seção 6.2.2. Somando os atrasos de tempo desses serviços, é possível obter o atraso de tempo da orquestração do experimento de controle como serviço.

A Figura 74 mostra a distribuição do histograma dos atrasos 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 105 ms e tem a presença de poucos valores discrepantes (*outliers*), o que significa que não há determinismo para a execução da malha de controle 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 74 - Histograma do Experimento de Controle com a Arquitetura MOA da Figura 71



Fonte: Autor.

Um cálculo estatístico foi feito a partir dos dados de atrasos visando a obtenção do atraso médio (Td) da comunicação e execução dos serviços para o experimento. A variabilidade dos valores de atraso de tempo fornece o *jitter* (J) na MOA desenvolvida. A Tabela 5 resume os resultados de tempo dos serviços e a execução da aplicação do experimento.

Os altos valores de *jitter* na Tabela 5 confirmam a grande variação do ciclo de controle de malha fechada (aplicativo de controle de processo) verificado na Figura 74. O termo “Pior Valor” corresponde aos valores mais altos (*outliers*) obtidos para cada tempo. No entanto,

verificou-se que eles correspondem a menos que 1% dos valores amostrados. A mesma conclusão também pode ser aplicada ao atraso de tempo da execução dos serviços. As comunicações do *Transporter* e *Gateway* nas respostas às solicitações de serviços são a razão da variabilidade.

Tabela 5 - Análise Temporal e de Desempenho de Comunicação do Experimento de Controle com a Arquitetura MOA da Figura 71

Descrição	Tempo de Atraso Médio (Td) [ms]	Jitter (J) [ms]	Pior Valor [ms]
Serviço DAQ: Ação de Aquisição do Sensor (PV)	24,5	31,5	655,2
Serviço Controle: Ação PIDPlus	21,8	17,3	77,8
Serviço DAQ: Ação de atualização do Atuador (MV)	43,0	21,0	120,6
Ciclo de Controle de Malha Fechada	105,5	43,9	785,9

7.2.2 Orquestração de Serviços usando Linguagem Ladder

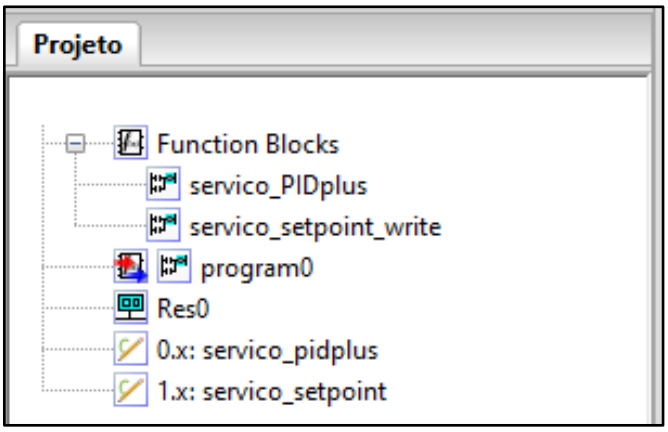
Desde 1968, quando o primeiro Controlador lógico programável (CLP) foi inventado, fornecedores desenvolveram seu próprio hardware proprietário e soluções de software voltadas para o ambiente industrial de forma fechada. Está forma fechada dificulta a integração com tecnologias mais atuais, como a Arquitetura Orientada a Serviços (SOA). Visando habilitar essa integração, este trabalho desenvolveu um mecanismo de programação de sistemas industriais baseados na arquitetura MOA deste trabalho utilizando as linguagens de programação de CLPs definidas pela IEC 61131-3: FBD (*Function Block Diagram*, Diagrama de Blocos), LD (*Ladder Diagram*, Diagrama Ladder), ST (*Structured Text*, Texto Estruturado), IL (*Instruction List*, Lista de Instruções) e SFC (*Sequential Function Chart*, Diagrama de Funções Sequenciais).

Para este desenvolvimento foi utilizado o software de editor de programas OpenPLC do projeto OpenPLC (OPENPLC, 2019). Este editor é compatível com as cinco linguagens de programação definidas pela IEC 61131-3. Blocos lógicos contendo códigos para realização de requisições via API REST da arquitetura MOA foram desenvolvidos para o OpenPLC Editor. Dessa forma, compatibilizou-se a comunicação entre os serviços da arquitetura MOA e os recursos de programação do ambiente de programação do OpenPLC Editor. Dessa forma, o ambiente OpenPLC Editor pode desenvolver uma Aplicação Externa para composição dos serviços disponíveis na arquitetura MOA.

Para apresentar este desenvolvimento, esta seção detalha o uso da Linguagem Ladder do OpenPLC Editor executando a orquestração de serviços para uma Aplicação de Controle de Processo. No entanto, seria possível o uso de qualquer uma das cinco linguagens proposta pela norma IEC 61131-3. Para o desenvolvimento dessa aplicação foi realizada a orquestração com

OpenPLC conforme apresentado na Figura 79. Através do OpenPLC Editor, que é uma IDE de desenvolvimento e programação de CLP compatível com a norma IEC 61131-3, o usuário pode programar a sua lógica de controle usando uma das 5 linguagens padronizadas. Este programa desenvolvido é compilado e carregado remotamente no servidor do OpenPLC para execução. Na aba de projetos do Editor do OpenPLC, na Figura 75, é mostrado onde estão os serviços PIDPlus e Setpoint que são funções em C que foram elaboradas e incluídas no projeto.

Figura 75 - Aba de Projeto do OpenPLC



Fonte: Autor.

A Figura 76 apresenta parte da função em C que foi incluída na biblioteca Ladder do Editor de programa do OpenPLC.

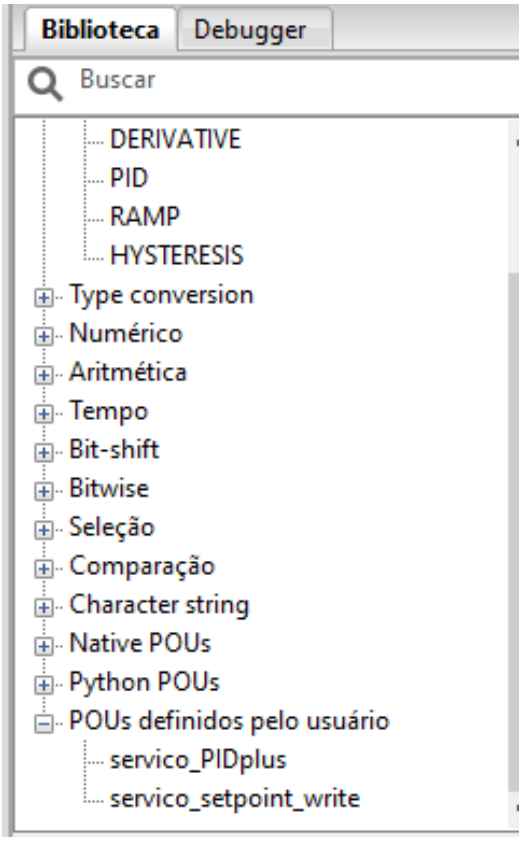
Figura 76 - Programa em C que foi incluído no projeto do OpenPLC

```
void __publish_0(void)
{
//S_mv=E_pv+E_setpoint+E_kp+E_ti+E_td;
/inicialiazação da requisição
// first what are we going to send and where are we going to send it?
    int portno =        3000;
    char *host =        "192.168.0.103";//IP do transpoter
    char *message_fmt_PIDplus = "GET /controle3/pidplus?pv=%d&ti=%d&setpoint=%d&td=%d&kp=%d HTTP/1.0\r\n\r\n";

struct hostent *server;
        struct sockaddr_in serv_addr;
        int sockfd, bytes, sent, received, total;
        char message[1024],response[4096];



        //fill in the parameters

        sprintf(message,message_fmt_PIDplus,E_pv,E_ti,E_setpoint,E_td,E_kp);
```

Fonte: Autor.

Como pode ser visto na Figura 77, as funções C que são criadas no projeto são incluídas na Biblioteca Ladder do Editor de programa do OpenPLC.

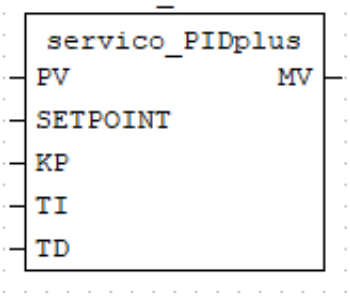
Figura 77 - Biblioteca de funções do OpenPLC



Fonte: Autor.

Depois de incluídas as funções em C na biblioteca do Editor de programa do OpenPLC estas funções podem ser usadas como blocos de funções no programa Ladder como apresentado na Figura 78.

Figura 78 - Bloco Ladder do serviço PIDPlus

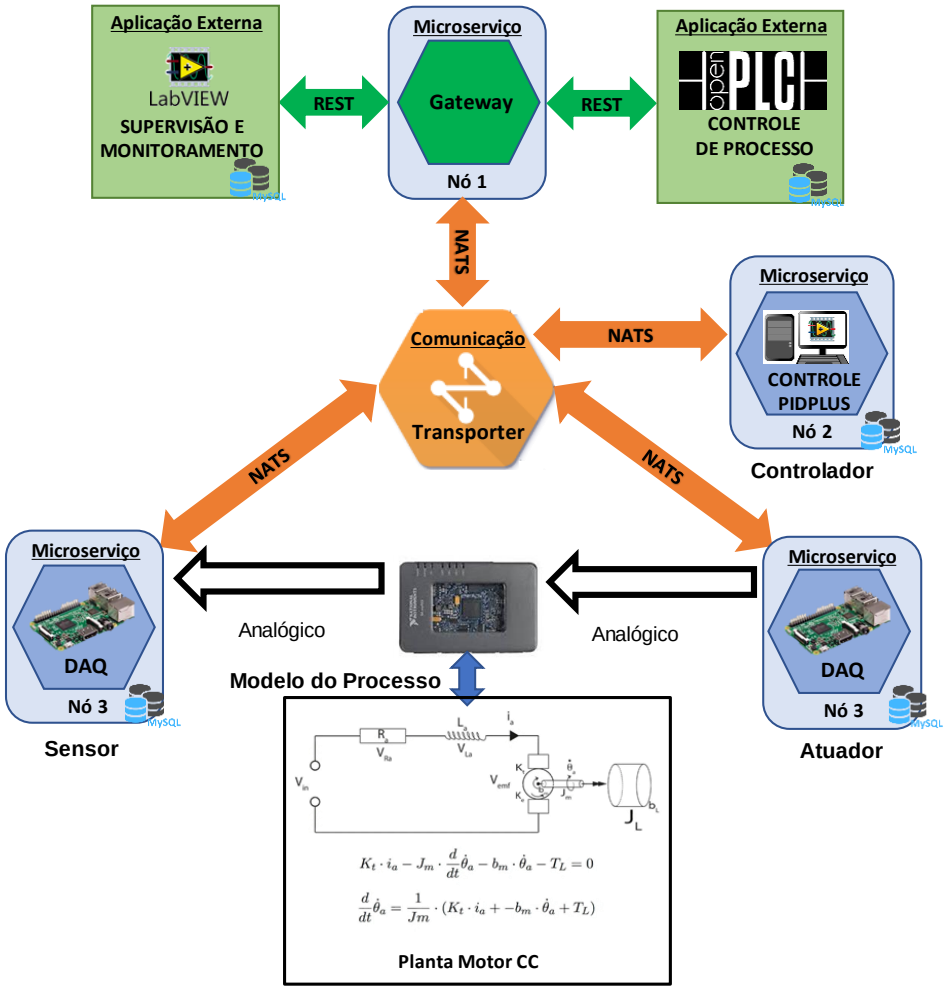


Fonte: Autor.

Experimento realizado

No segundo experimento realizado, a malha de controle de velocidade de motor CC em ambiente HIL foi utilizada com a arquitetura MOA. A estrutura geral desse experimento envolvendo a MOA é apresentada na Figura 79. Essa estrutura é semelhante ao experimento anterior da Figura 71, porém usando o software OpenPLC para orquestração dos serviços.

Figura 79 - Arquitetura MOA para o Experimento de Controle de Processo da Planta HIL com Orquestração de Serviços via OpenPLC

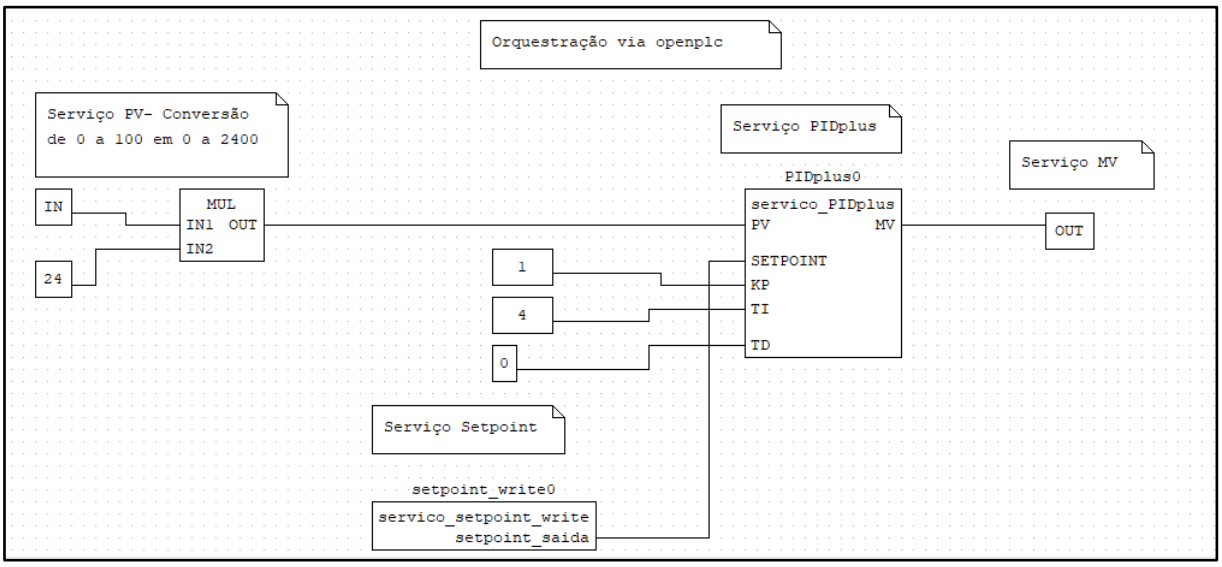


Fonte: Autor.

A Figura 80 apresenta a Linguagem Ladder do OpenPLC executando a sequência de orquestração de serviços para Controle de Processo. A estrutura usada no ensaio utiliza um processo simulado de um motor CC como descrito no tópico 7.1.2, o qual é executado no usando a técnica HIL. O serviço DAQ é usado para amostragem e atualização das variáveis do processo. A comunicação com serviço DAQ é executada através da camada de driver do OpenPLC por funções em linguagem C que enviam comandos REST do serviço API Gateway. A sequência de orquestração de serviços apresentado na Figura 80 é uma Aplicação Externa de Controle de Processo usando o serviço DAQ para amostragem e atualização das variáveis do processo. A comunicação e integração dos serviços são realizadas através dos blocos do OpenPLC, onde os blocos PIDPlus e SETPOINT são funções em C que enviam comandos REST do serviço API Gateway.

O microserviço DAQ utiliza duas ações: Aquisição de Entrada (PV) e Atualização de Saída (MV). O microserviço Controle PIDPlus utiliza as ações para recebimento das variáveis do controlador e transmissão da ação de controle calculada.

Figura 80 - Orquestração de Serviços usando Linguagem Ladder



Fonte: Autor.

Na orquestração de serviços executadas no OpenPLC, na qual cada microserviço está alocado num nó diferente, a cada tempo de ciclo ou atualização da malha de controle, a orquestração no OpenPLC manda uma requisição para o microserviço DAQ. O microserviço DAQ realiza a aquisição de dados de um canal de entrada e transmite esse dado de volta para a Aplicação Externa. A Aplicação Externa envia esse dado de entrada da variável de interesse, juntamente com os parâmetros de configuração do controlador, para o microserviço Controle PIDPlus através de uma nova requisição. O microserviço Controle PIDPlus executa o algoritmo de controle PIDPlus, e calcula um novo valor do sinal de controle para ser aplicado ao atuador da malha. O sinal de controle calculado é então transmitido de volta para a Aplicação Externa através de uma resposta.

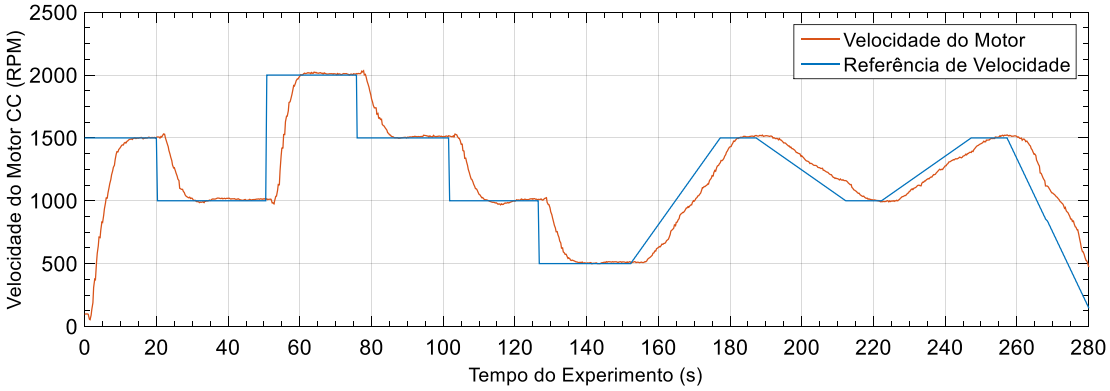
Finalmente, a Aplicação envia esse dado do sinal de controle para o microserviço DAQ, que executa a ação de atualizar uma saída ou canal de saída do hardware para atuação sobre a planta. Conforme informado, todo esse processo de orquestração é repetido ciclicamente de acordo com o tempo de ciclo definido para a malha de controle.

Testes operacionais utilizando orquestração via OpenPLC através da linguagem Ladder para o experimento de controle da malha de motor CC em ambiente HIL foram realizados. A Figura 81 apresenta os resultados do sistema de controle, armazenados pela Aplicação de Supervisão desenvolvida em LabVIEW, da curva de resposta em relação ao setpoint

estabelecido. Essa Aplicação de Supervisão teve de ser usada, pois o ambiente do OpenPLC Editor é somente um ambiente de programação, não apresentando interface para visualização/monitoramento de variáveis no tempo contínua. Devido a essa limitação do Open PLC Editor, não foi possível coletar os tempos de execução dos serviços e da orquestração da malha de controle conforme feito no experimento anterior.

A aplicação da arquitetura MOA com orquestração via programação Ladder nesse caso comprova a viabilidade de uso da arquitetura proposta em aplicações industriais. É importante verificar que o (re)uso dos serviços fornece modularidade e escalabilidade para a aplicações, flexibilizando a implantação de novas malhas de controle e padronizando o acesso às informações da planta através da linguagem Ladder.

Figura 81 - Resposta do Experimento de Controle de Processo usando orquestração via Ladder



Fonte: Autor.

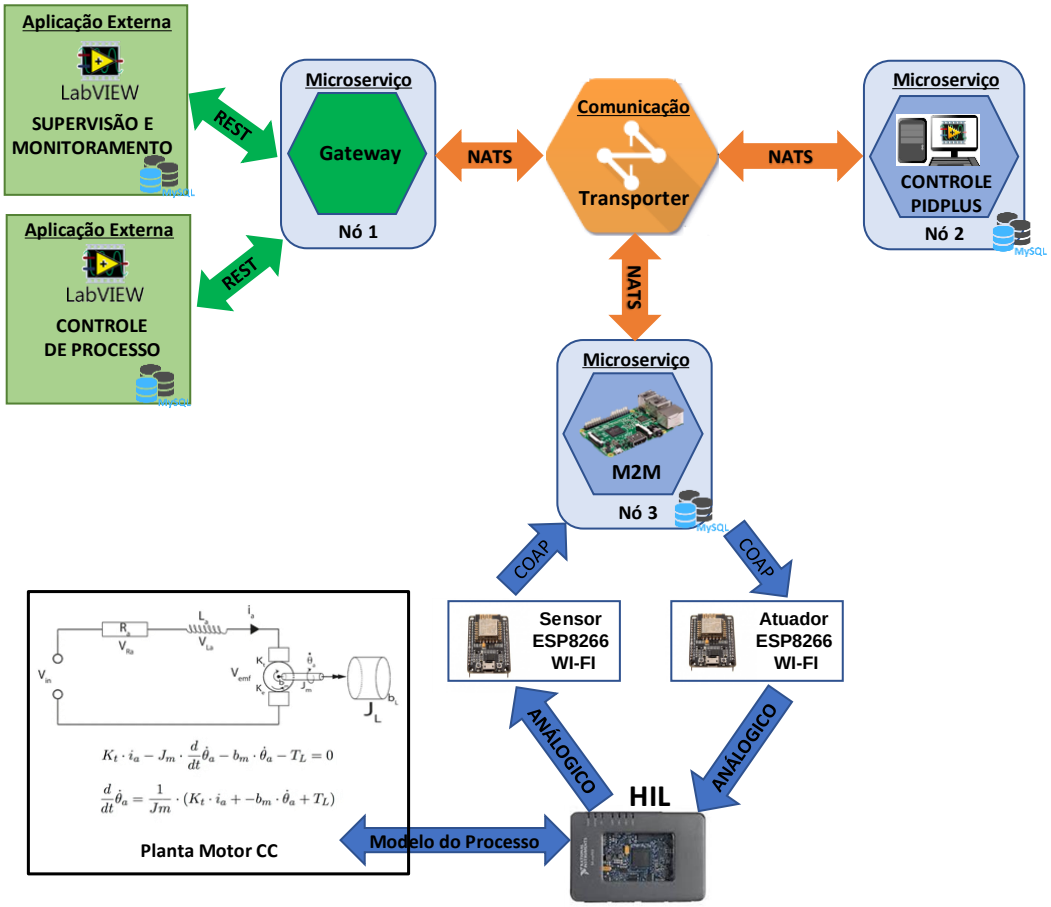
7.3 EXPERIMENTO 3: CONTROLE USANDO SERVIÇOS M2M E CONTROLE PIDPLUS

No terceiro experimento realizado, a malha de controle de velocidade de motor CC ambiente HIL foi utilizada com a arquitetura MOA. A estrutura geral desse experimento envolvendo a MOA é apresentada na Figura 82.

Nessa implementação, uma Aplicação Externa de “Controle de Processos” (seção 6.2.2) e uma de “Supervisão e Monitoramento” (seção 6.2.1) foram criadas a partir da orquestração dos microserviços de Controle PIDPlus, M2M, *Transporter* e Gateway. Essas aplicações foram desenvolvidas usando o software LabVIEW e são responsáveis pelo monitoramento (plotagem das curvas de resposta de saída), supervisão de variáveis e envio dos dados de controle do processo e por implementar o algoritmo de controle (PIDPlus) da malha. O serviço API *Gateway*, o serviço *Transporter* e as aplicações externas Controle e Supervisão são executados

em um computador (PC). O banco de dados utilizado pelos serviços para armazenamento dados das variáveis é o MySQL.

Figura 82 - Arquitetura MOA para o Experimento de Controle de Processo da Planta HIL usando Serviços M2M e Controle PIDPlus



Fonte: Autor

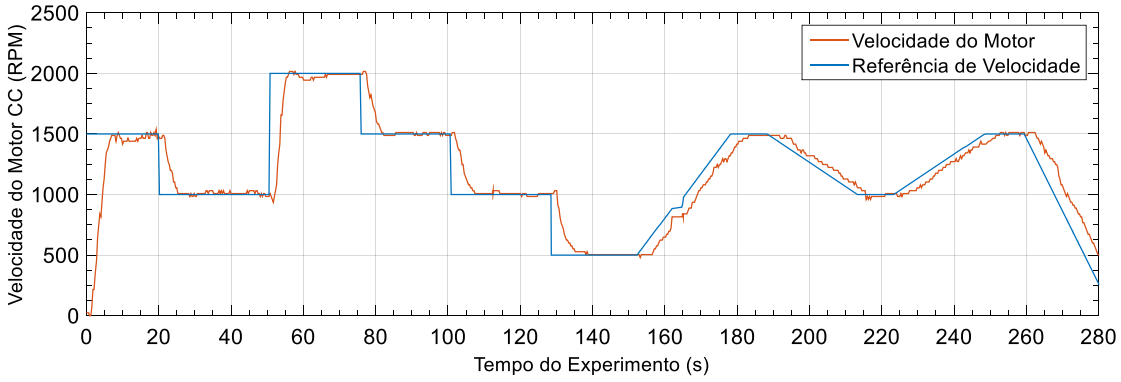
O serviço M2M (Nó 3) foi usado para aquisição da variável do processo (PV) e atualização do sinal de controle na planta (MV) usando o protocolo de IoT CoAP. O serviço M2M é responsável pela aquisição de dados remota do sensor e do atuador. Um sistema embarcado (Raspberry Pi 3B+ com distribuição Raspbian Linux) foi usado para a implementação e operação do serviço M2M. O Sensor e o Atuador consistem em módulos de hardware NodeMCU (NODEMCU, 2018) com o microcontrolador SoC (*System On a Chip*) ESP8266 (ESP8266, 2017).

O NodeMCU implementa o protocolo CoAP necessário para a comunicação com o serviço M2M (Nó 3). O bloco Sensor adquire (ADC) a variável de processo (PV) usando um sinal analógico de uma saída DAC do MyRIO. A planta do atuador é responsável por receber os sinais de controle calculados pelo serviço Controle PIDPlus (Nó 2). O atuador também usa um sinal analógico (PWM + filtro) para atuar na planta (MyRIO).

O serviço de Controle PIDPlus (Nó 2) implementa o algoritmo PIDPlus em *JavaScript* para o cálculo do controle de processo, acessando as variáveis do processo (amostragem do sensor e sinal de controle do atuador) via serviço M2M (Nó 3). O serviço *Transporter* utilizado para comunicação foi o NATS. O serviço *Gateway* foi usado para comunicação dos dados via REST dos serviços com a aplicação externa criada no LabVIEW.

Testes operacionais com a MOA proposta na estrutura da Figura 82 foram realizados. O controle da malha fechada foi definido com um período de ciclo de 300 ms (aquisição do sensor, computação do controle e atualização do sinal de controle na planta). A Figura 83 mostra os resultados da resposta do sistema de controle de velocidade do motor e a Figura 84 mostra o sinal de controle aplicado ao motor. Ambas as respostas foram armazenadas pela Aplicação de Supervisão.

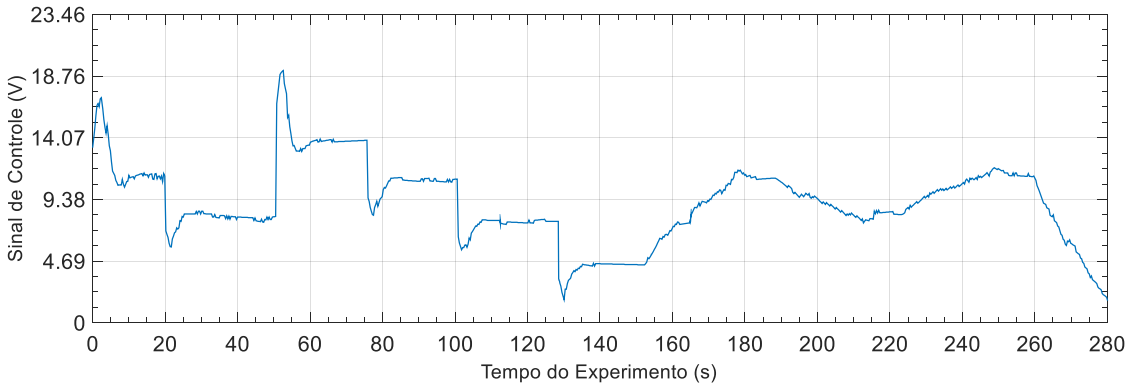
Figura 83 – Resposta do Experimento de Controle da Planta HIL com a Arquitetura MOA da Figura 82



Fonte: Autor.

A Figura 83 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 84 mostra o sinal de controle aplicado ao motor.

Figura 84 – Sinal de Controle do Experimento de Controle com a Arquitetura MOA da Figura 82

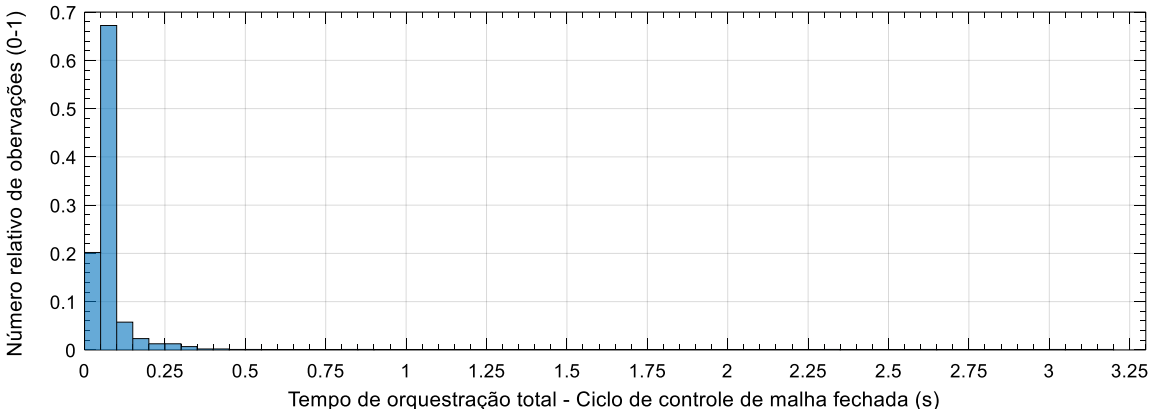


Fonte: Autor.

A análise temporal e de desempenho da comunicação em rede e da orquestração também foi realizada para o experimento da Figura 82. A Figura 85 mostra a distribuição do histograma dos atrasos 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 mais regular, se aproximando do valor médio de 91 ms (com quase 70% dos valores da distribuição), porém tem a presença de valores discrepantes (*outliers*). No entanto, o resultado demonstra melhor determinismo para a execução da malha de controle 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. Também é mostrado na Figura 85 poucas variações de medições de ida e volta das requisições no experimento realizado na ordem de 2 a 3 segundos gerados pelos módulos ESP8266 com o protocolo CoAP onde PIDPlus foi capaz de lidar com essas variações.

Figura 85 - Histograma do Experimento de Controle com a Arquitetura MOA da Figura 82



Fonte: Autor.

Um cálculo estatístico foi feito a partir dos dados de atrasos visando a obtenção do atraso médio (Td) da comunicação e execução dos serviços para o experimento. A variabilidade dos valores de atraso de tempo fornece o *jitter* (J) na MOA desenvolvida. A Tabela 6 resume os resultados de tempo dos serviços e a execução da aplicação do experimento.

Os altos valores de *jitter* na Tabela 6 confirmam a grande variação do ciclo de controle de malha fechada (aplicativo de controle de processo) verificado na Figura 85. O termo “Pior Valor” corresponde aos valores mais altos (*outliers*) obtidos para cada tempo. No entanto, verificou-se que eles correspondem a menos que 1% dos valores amostrados. A mesma conclusão também pode ser aplicada ao atraso de tempo da execução dos serviços. As

comunicações do *Transporter* e *Gateway* nas respostas às solicitações de serviços são a razão da variabilidade.

É possível verificar na Tabela 6 que no caso do serviço M2M existe um problema em relação à execução da ação de aquisição do sensor (PV), que impacta em altos valores para as estatísticas de tempos calculados. É importante informar que esse problema não está relacionado à execução do código do serviço M2M (JavaScript) e sim, na operação do módulo remoto de I/O com a comunicação CoAP. A implementação do código do módulo remoto (ESP8266) apresentou algumas inconsistências em relação à resposta da requisição CoAP recebida. Em alguns casos notava-se um "travamento" do módulo para retornar a requisição da variável de entrada (PV).

Como a ação de aquisição do sensor (PV) do serviço M2M aguarda a resposta do módulo remoto para retornar o valor via Transporter, verifica-se a presença desses grandes valores nos tempos medidos para a execução dessa ação. O mesmo não acontece com os tempos da execução da ação de atualização do atuador (MV) do serviço M2M, pois nesta ação o serviço M2M já retorna a confirmação de comando, sem a necessidade de aguardar a resposta do módulo remoto. Dessa forma, é necessária uma atualização desse código CoAP do módulo remoto desenvolvido ou utilizar outro módulo remoto comercial compatível com os protocolos de comunicação do serviço M2M (Modbus TCP/IP e CoAP).

Tabela 6 - Análise Temporal e de Desempenho de Comunicação do Experimento de Controle com a Arquitetura MOA da Figura 82

Descrição	Tempo de Atraso Médio (Td) [ms]	Jitter (J)[ms]	Pior Valor [ms]
Serviço M2M: Ação de Aquisição do Sensor (PV)	62,90	174,86	3095,80
Serviço Controle: Ação PIDPlus	8,11	7,17	58,95
Serviço M2M: Ação de atualização do Atuador (MV)	18,89	20,51	256,76
Ciclo de Controle de Malha Fechada	91,18	181,27	3183,71

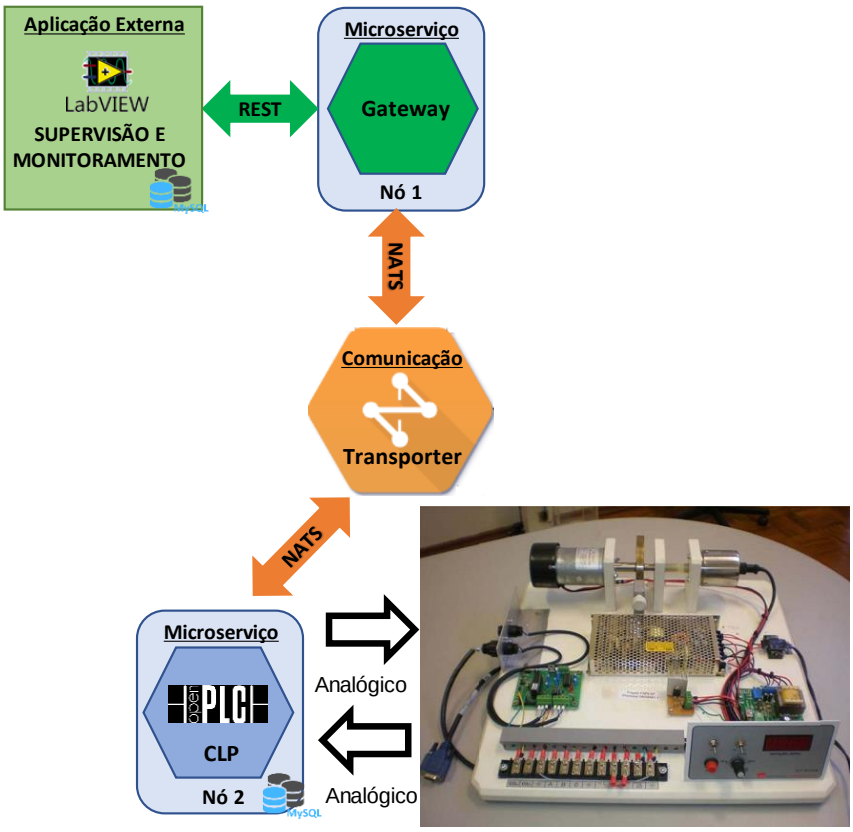
7.4 EXPERIMENTO 4: CONTROLE USANDO SERVIÇO CLP

No quarto experimento realizado, a malha de controle de velocidade de motor CC da planta didática foi utilizada com a arquitetura MOA. A estrutura geral desse experimento envolvendo a MOA é apresentada na Figura 86.

O microserviço CLP é responsável por integrar as funcionalidades de um controlador lógico programável (CLP) na arquitetura do *Moleculer*. Nessa implementação, uma Aplicação Externa de "Controle de Processos" (seção 6.1.3) e uma de "Supervisão e Monitoramento" (seção 6.2.1) foram criadas a partir da orquestração dos microserviços CLP, Transporter e

Gateway. Essas aplicações foram desenvolvidas usando o software LabVIEW e são responsáveis pelo monitoramento (plotagem das curvas de resposta de saída), supervisão de variáveis e envio dos dados de controle do processo.

Figura 86 - Arquitetura MOA para o Experimento de Controle de Processo da Planta Didática usando Serviço CLP



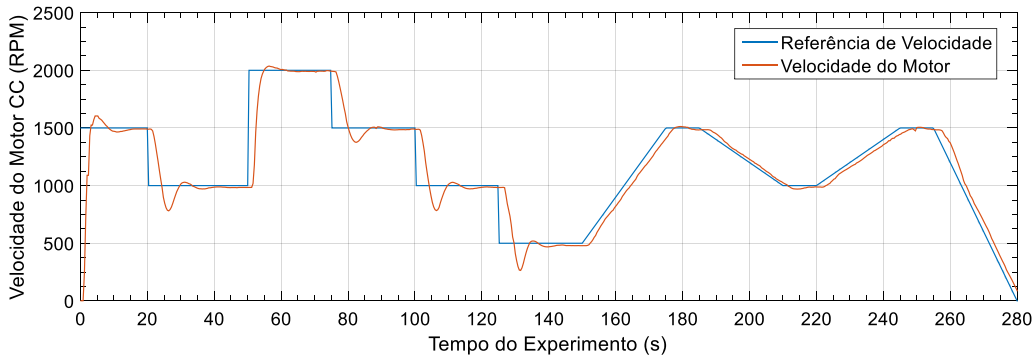
Fonte: Autor.

O microserviço CLP desenvolvido neste trabalho possui a estrutura mostrada na Figura 86. O Nó CLP foi desenvolvido para operar num sistema embarcado composto por uma Raspberry Pi 3B+ rodando Linux. O Nó CLP desenvolvido é composto pelo software do framework Moleculer e da codificação do Serviço CLP, que implementa as funcionalidades e ações do serviço (descritas na seção 5.1.1.1). Este microserviço fornece as funcionalidades de um Controlador Lógico Programável (CLP) com a vantagem de estar disponibilizado com um serviço. Para o desenvolvimento desse microserviço foi realizada a integração do projeto OpenPLC (OPENPLC, 2019) junto à arquitetura de microserviço.

Testes operacionais com a MOA proposta na estrutura da Figura 86 foram realizados. A Figura 87 mostra os resultados da resposta do sistema de controle de velocidade do motor e a

Figura 88 mostra o sinal de controle aplicado ao motor. Ambas as respostas foram armazenadas pelo aplicativo Supervisão.

Figura 87 - Resposta do Experimento de Controle da Planta Didática com a Arquitetura MOA da Figura 86

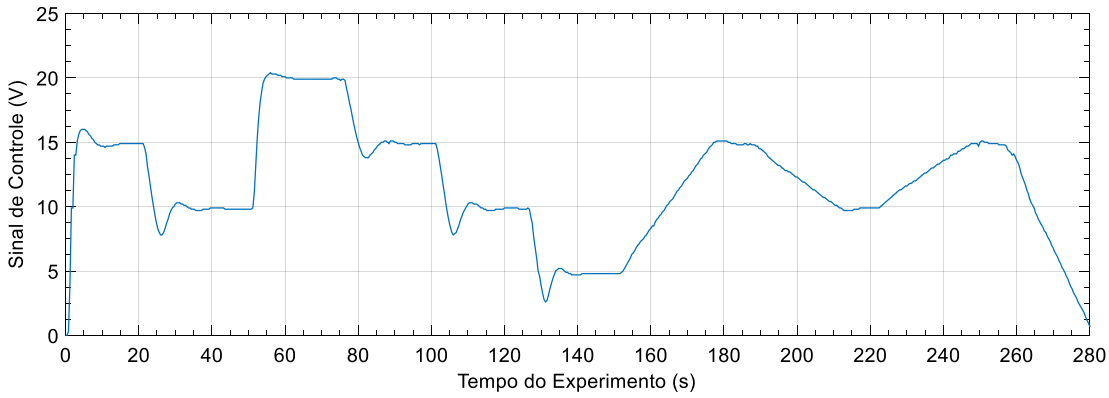


Fonte: Autor.

A Figura 88 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 sobretudo de implementação com controle lógico, foi possível verificar que a aplicação do controle usando o microserviço CLP é viável.

É importante verificar que com o uso do microserviço CLP, a execução e atualização da malha é feita localmente no sistema embarcado usando o tempo de ciclo padrão do OpenPLC. Da mesma forma que no experimento da seção 7.2.2, não foi possível coletar os tempos de execução dos serviços e da orquestração da malha de controle conforme feito no experimento anterior.

Figura 88 - Sinal de Controle do Experimento de Controle com a Arquitetura MOA da Figura 87



Fonte: Autor.

7.5 EXPERIMENTO 5: MÚLTIPLAS MALHAS DE CONTROLE E REPLICAÇÃO DE SERVIÇOS

No quinto experimento foi realizado a replicação de serviços para implementação de várias malhas de controle. Esse experimento é importante para analisar impacto da comunicação dos serviços no desempenho das malhas de controle. As malhas implementadas são: uma malha de controle de velocidade de motor CC da planta didática e quatro malhas do modelo dessa planta didática simulada usando a tecnologia HIL. A estrutura geral desse experimento envolvendo a MOA é apresentada na Figura 89.

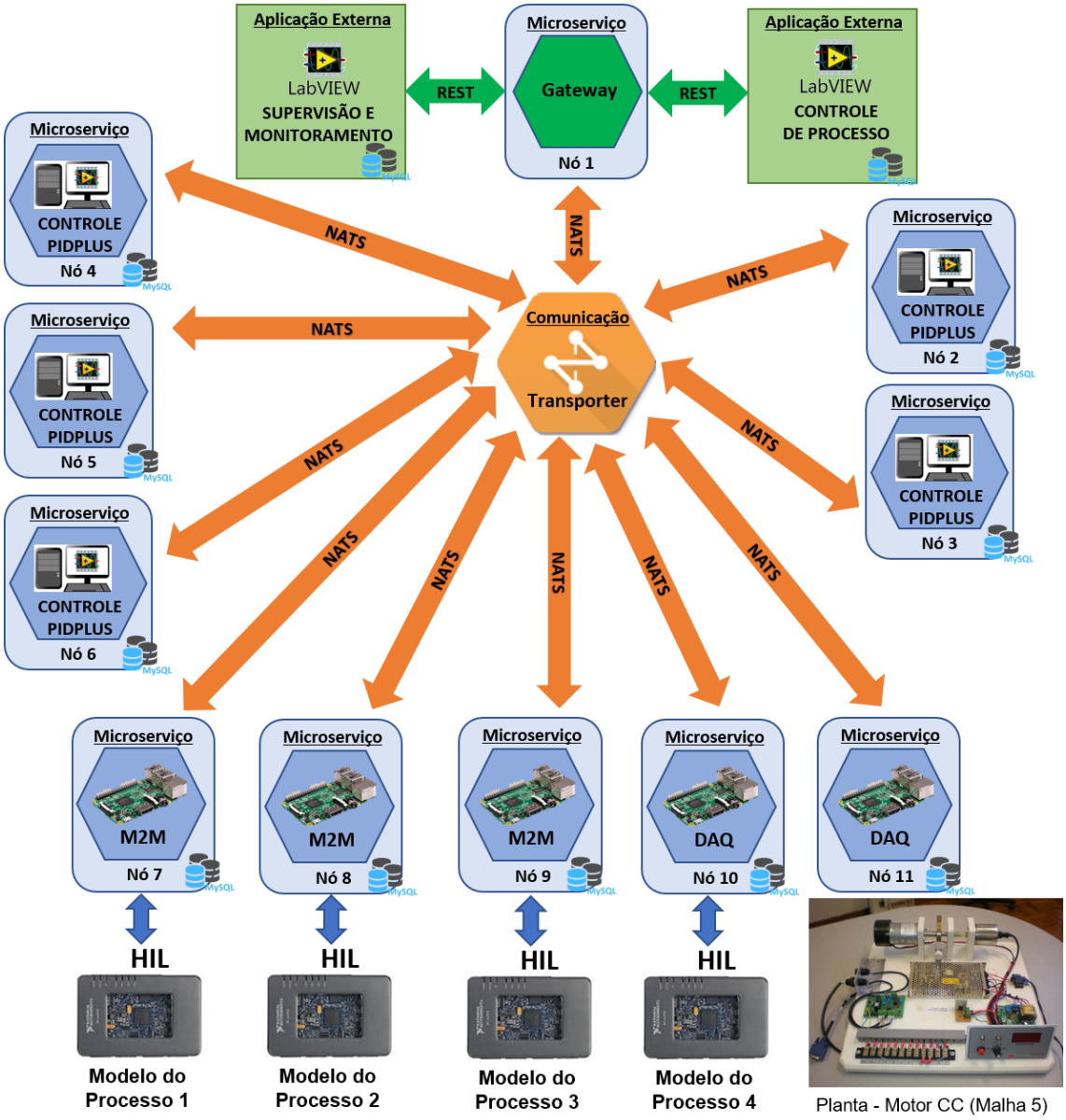
Nesse experimento, realizou-se a replicação dos microserviços desenvolvidos no trabalho para a implementação das outras malhas de controle. A replicação de microserviços facilitou e tornou o desenvolvimento e a implantação do sistema de controle com múltiplas malhas muito mais flexível e escalável. Esse é um dos pontos fortes da adoção da arquitetura MOA para aplicações de automação e controle. Como os microserviços utilizados (DAQ, M2M e Controle PIDPlus) já haviam sido desenvolvidos e testados durante o trabalho, para este experimento basicamente foi necessário a replicação, ou seja, cópia dos microserviços com códigos de 1 a 5 para cada malha criada, respectivamente, para organização.

Assim, como o experimento contemplou 5 malhas de controle, sendo 3 malhas usando os serviços M2M e Controle PIDPlus e outras 2 malhas usando os serviços DAQ e Controle PIDPlus, a replicação dos serviços ocorreu conforme mostrada na Figura 89. Dessa forma, através de acesso remoto a cada sistema embarcado ou a cada computador usado para rodar os nós do Moleculer, é bastante simples realizar a alocação de cada serviço em cada nó conforme a configuração/operação requerida para a arquitetura MOA.

1. Malha de controle 1: malha de controle do motor em ambiente HIL. Consiste na orquestração dos serviços M2M (renomeado como “M2M1.js”) corresponde ao nó 7, serviço Controle PIDPlus (renomeado para “PIDPlus1.js”) correspondente ao nó 2;
2. Malha de controle 2: malha de controle do motor em ambiente HIL. Consiste na orquestração dos serviços M2M (renomeado como “M2M2.js”) corresponde ao nó 8, serviço Controle PIDPlus (renomeado para “PIDPlus2.js”) correspondente ao nó 3;
3. Malha de controle 3: malha de controle do motor em ambiente HIL. Consiste na orquestração dos serviços M2M (renomeado como “M2M3.js”) corresponde ao nó 9, serviço Controle PIDPlus (renomeado para “PIDPlus3.js”) correspondente ao nó 4;
4. Malha de controle 4: malha de controle do motor em ambiente HIL. Consiste na orquestração dos serviços DAQ (renomeado como “DAQ4.js”) corresponde ao nó 10, serviço Controle PIDPlus (renomeado para “PIDPlus4.js”) correspondente ao nó 5;

5. Malha de controle 5: malha de controle do motor da planta didática. Consiste na orquestração dos serviços DAQ (renomeado como “DAQ5.js”) corresponde ao nó 11, serviço Controle PIDPlus (renomeado para “PIDPlus1.js”) correspondente ao nó 6;

Figura 89 - Arquitetura MOA para o Experimento de Controle de Múltiplas Malhas



Fonte: Autor.

Nesse experimento, uma Aplicação Externa de “Controle de Processos” (6.2.2) e uma de “Supervisão e Monitoramento” (6.2.1) foram criadas a partir da orquestração da replicação de cinco microserviços de Controle PIDPlus, três serviços M2M, dois serviços DAQ, Transporter e um serviço Gateway como pode ser visto na Figura 89. Essas aplicações foram desenvolvidas usando o software LabVIEW e são responsáveis pelo monitoramento (plotagem das curvas de resposta de saída), supervisão de variáveis e envio dos dados das malhas controle dos processos.

O serviço API Gateway, o serviço Transporter e as aplicações externas Controle e Supervisão são executados em um computador (PC). O serviço Transporter utilizado para comunicação foi o NATS. O serviço API Gateway foi usado para comunicação dos dados via REST dos serviços com a aplicação externa criada no LabVIEW.

Os serviços M2M (Nó 7, Nó 8 e Nó 9) foram usados para aquisição das variáveis dos processos (PV) e atualização dos sinais de controle nas plantas (MV) usando o protocolo de IoT CoAP. Os serviços M2M rodam num sistema embarcado (Raspberry Pi 3B+ com distribuição Raspbian Linux). Cada serviço M2M é responsável pela aquisição de dados remota do sensor e do atuador (NodeMCU). Os NodeMCU implementam o protocolo CoAP necessário para a comunicação com o serviço M2M (Nó 7, Nó 8 e Nó 9). O bloco Sensor adquire (ADC) a variável de processo (PV) usando um sinal analógico de uma saída DAC do MyRIO. A planta do atuador é responsável por receber os sinais de controle calculados pelo serviço Controle PIDPlus (Nó 2, Nó 3 e Nó 4). O atuador também usa um sinal analógico (PWM + filtro) para atuar na planta (MyRIO).

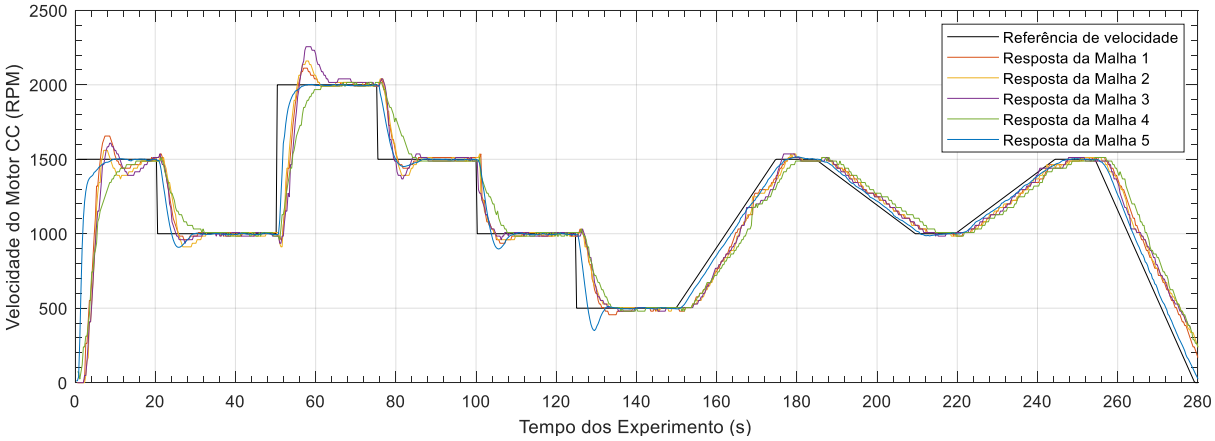
Os serviços DAQ (Nó 10 e Nó 11) realiza a aquisição da variável de processo (PV) e atualização da variável manipulada (MV) de duas plantas. O serviço DAQ é responsável pela aquisição de dados do sensor e do atuador. Um sistema embarcado (Raspberry Pi 3B com distribuição Raspbian Linux) foi usado para a implementação e operação dos serviços DAQ. O sensor adquire a variável de processo (PV) usando um sinal analógico. O atuador é responsável por receber os sinais de controle da aplicação de Controle (Nó 10 e Nó 11) e atuar na planta usando um canal PWM.

Os serviços de Controle PIDPlus (Nó 2, Nó 3, Nó 4, Nó 5 e Nó 6) implementam o algoritmo PIDPlus em JavaScript para o cálculo do controle de processo, acessando as variáveis do processo (amostragem do sensor e sinal de controle do atuador) via serviço M2M (Nó 7, Nó 8 e Nó 9) e DAQ (Nó 10 e Nó 11).

Testes operacionais com a operação das 5 malhas através da replicação de serviços da MOA proposta na estrutura da Figura 89 foram realizados. Os controles das malhas fechadas foram definidos com um período de ciclo de 300ms (aquisição do sensor, computação do controle e atualização do sinal de controle na planta).

A Figura 90 apresenta a comparação dos resultados das respostas dos sistemas de controle das malhas HIL e da malha de controle de motor CC e a Figura 91 mostram os sinais de controle aplicados, respectivamente. As respostas foram armazenadas pelo aplicativo Supervisão.

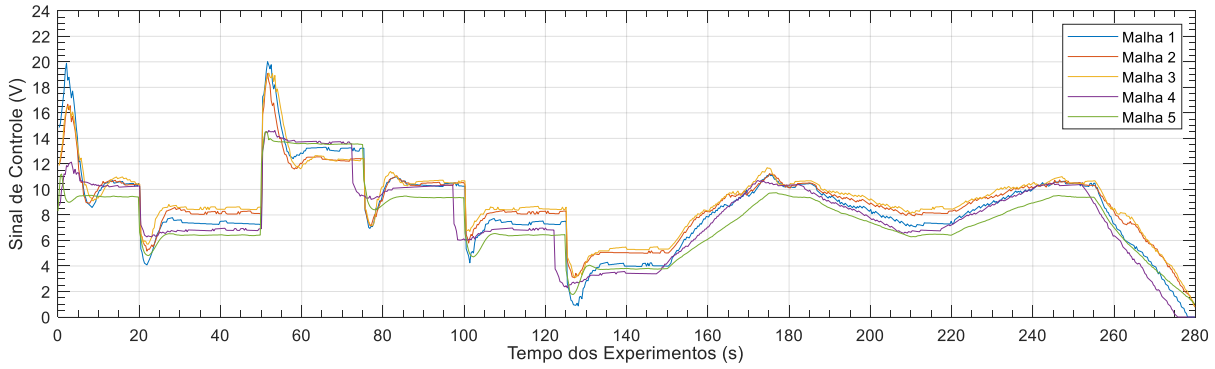
Figura 90 – Comparação das Respostas do Experimento com Múltiplas Malhas com a Arquitetura MOA da Figura 89



Fonte: Autor.

Na Figura 90 mostrados que os sistemas de controle são estáveis e controlados, pois os controles de velocidade das malhas HIL e da planta didática foram capazes de rastrear o ponto de ajuste definido, considerando as entradas de degrau e rampa. A Figura 91 mostra a comparação dos sinais de controle aplicado ao motor em cada malha. Não há saturação de controle. Os resultados obtidos com este experimento de múltiplas malhas com uso da replicação dos serviços da MOA são viáveis e demonstram a flexibilidade da adoção dessa arquitetura para aplicações de automação e controle com foco na I4.0.

Figura 91 - Comparação dos Sinais de Controle do Experimento com Múltiplas Malhas com a Arquitetura MOA da Figura 89



Fonte: Autor.

Os cálculos estatísticos para as múltiplas malhas de controle no experimento com replicação de serviços da Moa mostrada na Figura 89 foram feitos a partir dos dados de atrasos visando a obtenção do atraso médio (Td) da comunicação e execução dos serviços para os experimentos. Optou-se por não apresentar os gráficos de histograma como forma de simplificação e pelo fato da Tabela 7 contemplar as informações necessárias. A variabilidade

dos valores de atraso de tempo fornece o *jitter* (J) na MOA desenvolvida referente as cinco malhas testadas. O termo “Pior Valor” corresponde aos valores mais altos (outliers) obtidos para cada tempo, os quais novamente representaram menos de 1% dos valores amostrados. A Tabela 7 resume os resultados de tempo dos serviços e a execução das aplicações dos experimentos.

Tabela 7 - Análise Temporal e de Desempenho de Comunicação dos Experimentos de Controle com múltiplas malhas e replicação de serviços com a Arquitetura MOA da Figura 89

Malhas	Descrição	Tempo de Atraso Médio (Td) [ms]	Jitter (J) [ms]	Pior Valor [ms]
Malha 1	Serviço M2M: Ação de Aquisição do Sensor (PV)	84,20	181,01	2947,75
	Serviço Controle: Ação PIDPlus	6,89	6,07	147,86
	Serviço M2M: Ação de atualização do Atuador (MV)	11,97	18,31	278,74
	Ciclo de Controle de Malha Fechada	103,47	183,26	2961,74
Malha 2	Serviço M2M: Ação de Aquisição do Sensor (PV)	75,36	181,33	2908,79
	Serviço Controle: Ação PIDPlus	7,62	6,92	170,84
	Serviço M2M: Ação de atualização do Atuador (MV)	20,07	22,53	310,71
	Ciclo de Controle de Malha Fechada	103,12	184,85	2939,76
Malha 3	Serviço M2M: Ação de Aquisição do Sensor (PV)	123,76	210,53	2940,76
	Serviço Controle: Ação PIDPlus	6,64	4,36	40,96
	Serviço M2M: Ação de atualização do Atuador (MV)	10,94	14,71	215,79
	Ciclo de Controle de Malha Fechada	141,64	212,56	2953,74
Malha 4	Serviço DAQ: Ação de Aquisição do Sensor (PV)	54,99	181,89	2917,77
	Serviço Controle: Ação PIDPlus	10,65	5,92	36,97
	Serviço DAQ: Ação de atualização do Atuador (MV)	17,89	24,82	312,71
	Ciclo de Controle de Malha Fechada	83,90	186,73	2945,75
Malha 5	Serviço DAQ: Ação de Aquisição do Sensor (PV)	30,49	21,43	515,62
	Serviço Controle: Ação PIDPlus	41,38	27,52	338,13
	Serviço DAQ: Ação de atualização do Atuador (MV)	34,30	33,67	498,64
	Ciclo de Controle de Malha Fechada	128,38	55,69	646,31

7.6 COMPARAÇÃO E ANÁLISE DOS RESULTADOS

Considerando os resultados apresentados nos experimentos realizados com a operação da MOA em variadas configurações, analisou-se o impacto geral da comunicação entre os serviços e da orquestração dos serviços nas aplicações criadas no desempenho de cada malha de controle de forma individual. Conforme verificado nos gráficos de resposta de todas as malhas na Figura 90, a aplicação da MOA se mostrou confiável e eficiente, conseguindo controlar as variáveis e manter todos os processos estáveis.

Apesar de ser perceptível um leve aumento nos tempos de atraso médio (Td) e nos jitter (J) de execução dos serviços e de execução das aplicações de orquestração das malhas de

controle, quando comparado os dados das Tabela 5 a Tabela 7, esses incrementos não impactaram significativamente o desempenho de controle individuais das malhas, as quais apresentaram desempenho semelhante tanto na operação dedicada (somente uma malha nas seções 7.2 e 7.3) quanto na operação com múltipla malhas (seção 7.5). Verificou-se através da comparação dos dados das Tabela 5 a Tabela 7 que a comunicação em rede na MOA, considerando o protocolo NATS do serviço *Transporter* e do protocolo REST do serviço *API Gateway*, é sensível à variação da quantidade de nós (aumento ou diminuição de malhas) em operação na arquitetura.

A existência de uma maior quantidade de nós e microserviços disponíveis para uso acarreta um aumento no tempo requisição e execução dos serviços e nos tempos de orquestração das aplicações criadas. Em contrapartida, apesar de haver essa variabilidade no tempo requisição e execução dos serviços e nos tempos de orquestração das aplicações, a existência de uma maior quantidade de nós e microserviços disponíveis não impactou na confiabilidade da comunicação em rede da MOA. A quantidade de valores fora do normal (*outliers*) dos tempos citados continuou sendo bastante pequena e representando menos de 1% do total de requisições realizadas nos experimentos. Além disso, a quantidade de erros de comunicação (ou requisições sem resposta) foi irrisória ou descartável nos experimentos realizados.

Também pode ser verificado através das análises dos tempos dos experimentos apresentados na Tabela 7, que o tipo de arquitetura usada na MOA e a alocação dos microserviços e nós nas plataformas usadas para operação (computador e sistema embarcado) exercem relevante impacto no desempenho da comunicação, principalmente no caso da orquestração de serviços (foco deste trabalho). Neste trabalho, a arquitetura distribuída onde cada microserviço é alocado em um nó diferente foi utilizada. Isso significa que, apesar de ganhar em modularidade e resiliência, toda a comunicação entre serviços acontece através do serviço *Transporter*. No entanto, ainda que os microserviços estejam alocados em nós diferentes, a alocação conjunta, ou seja, na mesma plataforma de operação, de mais de um nó/serviço da arquitetura MOA impacta na redução do tempo de comunicação.

Isso acontece porque, mesmo os microserviços estando em nós diferentes e tendo que se comunicar via serviço *Transporter*, essa comunicação depende localidade da plataforma onde esses nós estão rodando. Quando microserviços rodam em plataformas diferentes, por exemplo cada microserviço/nó rodando em um computador diferente na infraestrutura de rede, toda a comunicação deve passar por algum dispositivo de rede (roteador em redes sem fio e switch em redes cabeadas). Dessa forma, ainda que a comunicação aconteça de forma transparente usando o *Transporter*, ela necessariamente inclui os tempos de transmissão do dado no meio físico e processamento pelo dispositivo de rede. Já quando mais de um microserviço roda na mesma

plataforma, principalmente no caso de algum serviço rodar juntamente com os serviços Transporter e/ou API Gateway, a comunicação entre esses serviços acontece em menor tempo. Isso acontece, pois nessas comunicações não há a necessidade de transmissão do dado no meio físico e processamento pelo dispositivo de rede pelo fato de ambos os nós estarem rodando na mesma plataforma.

Essa constatação pode ser verificada através da comparação dos resultados do experimento 5 (seção 7.5). No experimento 5, os nós/serviços Controle PIDPlus das malhas 1 a 4 estão alocados no mesmo computador dos serviços *Transporter* e API Gateway. Já o nó/serviço Controle PIDPlus da malha 5 está alocado em outro computador. Dessa forma, verifica-se que o tempo médio de requisição/execução dos serviços Controle PIDPlus das malhas 1 a 4 são menores, em torno de 9 ms, que o tempo médio de execução da malha 5, que apresenta valor de 41,38 ms. No caso do serviço Controle PIDPlus da malha 5, toda a comunicação via Transporter e via API Gateway (Aplicação Externa) precisa passar pelo roteador da rede sem fio utilizada. No caso das malhas 1 a 4, pelo fato de os todos os nós/serviços estarem no mesmo computador, esses tempos são menores.

Outros fatores foram constatados durante os experimentos, apesar de não terem sido analisados de forma mais contundente. Os atrasos da comunicação em rede na MOA relacionados tempo requisição e de orquestração das aplicações sofrem influência do meio físico de comunicação usado, no caso rede Ethernet TCP/IP cabeada ou rede Wi-Fi sem fio. Além disso, existe uma variação em relação ao tempo de execução dos serviços em um computador (nesse caso rodando Windows) e em um sistema embarcado (nesse caso uma Raspberry Pi 3B+ rodando Linux). Apesar do maior poder de processamento do computador em relação ao sistema embarcado, o tempo de execução de um serviço no computador tende a apresentar maior variabilidade. Esse fato deve estar relacionado à execução concorrente no sistema operacional Windows de diversos outros programas/serviços, de forma que a alocação de recursos computacionais para a execução do serviço seja variável.

As análises temporais e da comunicação em rede nos experimentos realizados comprovaram que, apesar do tempo de atraso (latência) e do *jitter* não serem desprezíveis, a arquitetura MOA é confiável e recomendada ou adequada para aplicações de automação e controle com orquestração de serviços (DAQ, M2M e Controle PIDPlus) com tempo de ciclo de ao menos 300 ms. Esse valor mínimo de tempo de ciclo considera a soma do valor do tempo médio de atraso (Td) mais o valor do *jitter* (J) relacionados à orquestração das aplicações criadas neste trabalho.

Ainda que no caso da malha 3 com serviço M2M do experimento com múltiplas malhas, esse valor definido não seja cumprido, cabe lembrar que a execução do serviço M2M na ação

de aquisição do sensor apresentou um problema de execução (explicado na seção 7.3) que acarreta em aumento dos valores do tempo de atraso e *jitter*. Caso esses altos valores (outliers) fossem removidos, por exemplo com a implementação de uma política de limite de tempo (*timeout)* no aguardo do retorno das requisições, haveria uma diminuição destes valores, acarretando uma diminuição dos tempos de ciclo com o uso do serviço M2M e consequentemente cumprindo a regra definida. Esse valor definido de 300 ms não se aplica para os casos de aplicações usando o serviço CLP, pois nesse serviço o tempo de ciclo (leitura do sensor, cálculo do controle e atualização da saída) segue o padrão do OpenPLC, que possui o valor de 50 ms.

O desenvolvimento de cada serviço em um nó facilita a replicação e reuso desses serviços em outras aplicações. Conforme explicado no experimento com múltiplas malhas, o procedimento de replicação de serviços é bastante simples. A redundância de serviços, ainda que não contemplada nos experimentos realizados, também representa outra importante característica do uso da arquitetura MOA para aplicações de automação e controle. A redundância de serviços pode ser facilmente implementada através da replicação (cópia) do mesmo serviço, mantendo-se o mesmo nome do serviço (sem alteração do nome). Nesse caso, a arquitetura MOA do Moleculer automaticamente verifica que existem duas ou mais instâncias (redundantes) do mesmo serviço, em caso em requisição desse serviço, ela automaticamente seleciona a melhor dessas instâncias para execução do mesmo.

8 CONCLUSÕES

Este trabalho apresentou uma revisão sobre SOA e sua aplicação industrial, discutindo as características e vantagens de uma proposta de arquitetura de automação e controle orientada a microserviços para aplicações no contexto de IIoT e I4.0. A revisão bibliográfica demonstrou que existe uma grande quantidade de pesquisas e desenvolvimento de SOA em aplicações industriais usando diferentes tecnologias de comunicação e integração. No entanto, ainda há uma carência relevante de desenvolvimentos de arquiteturas orientadas a microserviços para aplicações industriais, principalmente em relação a uma análise mais detalhada de desempenho da comunicação e execução de serviços e aplicações (composição de serviços).

A integração das tecnologias de automação e informática industrial numa arquitetura MOA permitiu o uso e compartilhamento de microserviços para obtenção de uma arquitetura flexível, escalável, interoperável, distribuída e conectada em rede. A arquitetura desenvolvida neste trabalho representa uma mudança de paradigma nas interações entre os diferentes sistemas industriais, equipamentos, aplicações e usuários. Apesar da estrutura hierárquica tradicional da arquitetura ISA95 ainda ser presente na indústria, a MOA desenvolvida representa uma arquitetura alternativa e complementar de informações que disponibiliza uma grande variedade de serviços, permitindo que as informações provenientes de sistemas heterogêneos possam ser obtidas de forma transparente ao usuário.

O *framework Moleculer* para microserviços foi escolhido para o desenvolvimento da arquitetura MOA, permitindo agilizar, organizar e estruturar o desenvolvimento e implantação das aplicações. Uma grande vantagem do uso desse *framework* é que todos os microserviços desenvolvidos possuem disponível um recurso automático de registro e descoberta, o qual permite que esses microserviços possam ser endereçadas e requisitados. Essa escolha se mostrou vantajosa, ao invés de criar toda a estrutura da SOA com visto em trabalhos na literatura, pois reduziu o tempo de desenvolvimento da MOA e permitiu que o foco fosse dado ao desenvolvimento dos serviços e aplicações para aplicações industriais.

A arquitetura MOA desenvolvida é composta de diferentes serviços e aplicações (composição de serviços) com foco na automação e controle de processos. Os serviços criados (comunicação M2M, CLP, DAQ e Controle PIDPlus) podem ser utilizados na MOA para a criação de aplicações a partir da Orquestração ou da Coreografia de serviços. Este trabalho focou no estudo detalhado do mecanismo de orquestração. A análise das aplicações desenvolvidas no trabalho e testadas nos experimentos realizados demonstrou a vantagem proporcionada pela composição de serviços na MOA para aplicações de automação e controle para a I4.0. As inúmeras possibilidades de composição de serviços fornecem enorme

flexibilidade para a criação das aplicações, diante da possibilidade de reuso dos serviços e composições.

Para comprovar a viabilidade de aplicação e validar o desenvolvimento da MOA, diversos experimentos com diferentes malhas de controle de processo foram apresentados. Os resultados desses experimentos demonstraram a confiabilidade e eficiência da arquitetura MOA, conseguindo controlar e manter todas as malhas de controle estáveis. Além das aplicações de controle, verificou-se que a implementação de aplicações de supervisão e monitoramento são significativamente simplificadas e facilitadas. As análises temporais e da comunicação em rede nos experimentos realizados comprovaram que, apesar da latência e do jitter serem significativos, a arquitetura MOA é confiável e recomendada para aplicações de automação e controle com orquestração de serviços com tempo de ciclo de ao menos 300 ms.

A comunicação transparente e padronizada entre serviços e aplicações proporcionada pela MOA foi um ponto forte dessa arquitetura. Esta funcionalidade proporciona interoperabilidade total e supera problemas de integração vertical de dados entre diferentes níveis hierárquicos das aplicações industriais. No entanto, constatou-se que a variação da quantidade de nós e microserviços disponíveis para uso acarreta um aumento no tempo requisição e execução dos serviços e nos tempos de orquestração das aplicações criadas. Além disso, a alocação física e lógica dos microserviços e nós nas plataformas de operação da MOA também impacta no desempenho da comunicação, principalmente no caso da orquestração de serviços.

A modularidade e escalabilidade da arquitetura MOA foram comprovadas através dos experimentos realizados. O desenvolvimento de cada serviço em um nó facilita a replicação e reuso desses serviços em outras aplicações. As aplicações são facilmente escaláveis e alteradas a partir do momento que as aplicações e serviços são desacoplados, ou seja, independentes uns em relação aos outros. Um ponto importante demonstrado é que a MOA pode ser facilmente expandida, por meio da criação de novos microserviços e aplicações para permitir a realização de novas funcionalidades.

O desenvolvimento da arquitetura MOA desse trabalho abre novas oportunidades de pesquisa para futuros trabalhos. O estudo da composição de serviços por Coreografia certamente deve ser realizado e seu desempenho comparado com os da orquestração deste trabalho. A criação de novos serviços e aplicações deve ser contemplada para flexibilizar o leque de uso da arquitetura. Um ponto importante para a análise será a implementação de mecanismos de segurança na arquitetura, indispensáveis para aplicações industriais. Além disso, a padronização total da comunicação e interoperabilidade fornecida permitirão a criação de novas aplicações no contexto de IIoT e I4.0, como realidade virtual e aumentada e gêmeos digitais.

9 REFERÊNCIAS BIBLIOGRÁFICAS

A VOZ DA INDÚSTRIA, ***Ebook:*** **Manufatura Avançada**. Advantech. Enabling an Intelligent Planet, 2016. Disponível em: http://www.advantech.com.br/. Acesso em: Janeiro, 2018.

ADOLPHS, P.; AUER, S.; BEDENBENDER, H.; BILLMANN, M.; HANKEL, M.; HEIDEL, R.; HOFFMEISTER, M.; HUHLE, H.; JOCHEM, M. **Structure of the Administration Shell:** Continuation of the Development of the Reference Model for the Industrie 4.0 Component. Plattform Industrie 4.0, 2016. Disponível em: https://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/structure-of-the-administration-shell.html. Acesso em: Janeiro, 2019.

AITOUFKIR, Z. **What Are RESTful Web Services?**. Iticsys Blog. Disponível em: http://iticsys.com/blog/2016/01/05/what-are-restful-web-services. Acesso em: Janeiro, 2019.

ARROWHEAD. The Arrowhead Framework, 2014. Disponível em: https://www.arrowhead.eu/. Acesso em: Abril, 2019.

ARROWHEAD TOOLS. Arrowhead Tools for Engineering of Digitalisation Solutions, 2019. Disponível em: https://cordis.europa.eu/project/id/826452. Acesso em: Abril, 2019.

BANGEMANN T, KARNOUSKOS S, CAMP R, CARLSSON O, RIEDL M, MCLEOD S, HARRISON R, COLOMBO AW, STLUKA P. **State of the art in industrial automation**. Industrial cloud-based cyber-physical systems Springer International Publishing, Switzerland, ch 2, 2014, pp. 23–47.

BAUER, M.; BOUSSARD, M.; BUI, N.; DE LOOF, J.; MAGERKURTH, C.; MEISSNER, S.; NETTSTRÄTER, A.; STEFA, J.; THOMA, M.; WALEWSKI, J. W. **IoT Reference Architecture**. In: Enabling Things to Talk. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013a. p. 163–211.

BAUER, M.; BUI, N.; LOOF, J. de; MAGERKURTH, C.; NETTSTRÄTER, A.; STEFA, J.; WALEWSKI, J. W. IoT Reference Model. In: BASSI, A.; BAUER, M.; FIEDLER, M.; KRAMP, T.; KRANENBURG, R. VAN; LANGE, S.; MEISSNER, S. (Ed.). **Enabling Things to Talk**. Berlin: Springer Berlin Heidelberg, 2013b. p. 113–162.

BELTRAN V., MARTINEZ J. A., SKARMETA A., An ARM- **Compliant IoT Platform**: Security by Design for the Smart Home. In: IEEE 5th Global Conference on Consumer Electronics, 2016.

BERGER, R. **Industry 4.0** – The new industrial revolution. Strategy Consultants, 2014. Disponível em: https://www.rolandberger.com/media/pdf/Roland_Berger_TAB_Industry_4_0_20140403.pdf. Acesso em: Novembro, 2015.

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: XXII Congresso Brasileiro de Automática, 2018, João Pessoa. Anais do CBA 2018. Campinas: SBA, 2018.

_____. **IoT-based Networked Control Systems**: A Proposal and Case Study. In: Networked Control Systems: Research Challenges and Advances for Application. 1ed.Hauppauge: Nova Science Publishers, 2018, v. 7, p. 171-196.

BIGHETI, J.A.; CALDIERI, M.R.; GODOY, E.P. **Automação e controle de processos na nuvem**: proposta e estudo de caso. In: Congresso Brasileiro de Automática (CBA), Vitória, ES, 2018.

BLEVINS, T; NIXON, M. AND WOJSZNIS, W. **PID Control Using Wireless Measurements**. American Control Conference (ACC), June 4-6, 2014. p. 1-6.

BLOMSTEDT, F.; FERREIRA, L.L.; KLISICS, M.; CHRYSOULAS, C. DE SORIA, I.M.; MORIN, B.; ZABASTA, A.; ELIASSON, J.; JOHANSSON, M.; VARGA, P. **The arrowhead approach for SOA application development and documentation**. IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society, Dallas, TX, 2014, pp. 2631-2637.

BOHN, H., BOBEK, A.; GOLATOWSKI, F. **SIRENA – Service Infrastructure for Real-time Embedded Networked Devices**: A service-oriented framework for different domains. Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, Morne, Mauritius, 2006.

BORMANN, C. (2014). COAP - **Constrained Application Protocol**. Disponível em: http://coap.technology/. Acesso em: Janeiro, 2017.

BORMANN, C.; SHELBY, Z. **Block-Wise Transfers in the Constrained Application Protocol (CoAP)**, 2016. Disponível em: https://tools.ietf.org/html/rfc7959. Acesso em: Janeiro, 2017.

CACKOVIC, V.; POPOVIC, Ž. **Management in M2M networks**. In: 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). Opatija, Croácia: IEEE, 2014, p. 501-506.

CANNATA, A.; GEROSA, M.; TAISCH, M. **SOCRADES:** A framework for developing intelligent systems in manufacturing. IEEE International Conference on Industrial Engineering and Engineering Management, Singapore, 2008, pp. 1904-1908.

CHEN Y.; DU, Z.; GARCIAACOSTA, M. **Robot as a service in cloud computing**. In Proc. IEEE Int. Symp. Serv. Oriented Syst. Eng., Jun. 2010, pp. 151–158.

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.

CISCO**. CISCO**: The Internet of Things Reference Model. Disponível em: http://cdn.iotwf.com/resources/71/IoT_Reference_Model_White_Paper_June_4_2014.pdf Acesso em: Janeiro, 2019

COLOMBO, A. W, KARNOUSKOS, S. BANGEMANN, T. **Towards the Next Generation of Industrial Cyber-Physical Systems**. In: Bangemann T, Karnouskos S, Delsing J, Stluka P, Harrison R, Jammes F, Lastra JL (eds), Ch 1, Industrial cloud-based cyber-physical systems. Springer International Publishing, Switzerland, 2014, pp 01–22.

COLOMBO, A. W, KARNOUSKOS, S., O. KAYNAK, Y. SHI, S. YIN. **Industrial Cyberphysical Systems**: A Backbone of the Fourth Industrial Revolution. In: IEEE Industrial Electronics Magazine, March 2017.

DECOTIGNIE, J. D. **Ethernet-based real-time and industrial communications**. Proceedings of the IEEE, 2005, v. 93, n. 6, pp. 1102-1117.

DELSING, J. **IoT Automation Arrowhead Framework**. CRC Press, 440 p., 2017.

_____. **Local Cloud Internet of Things Automation:** Technology and Business Model Features of Distributed Internet of Things Automation Solutions. IEEE Industrial Electronics Magazine, vol. 11, no. 4, pp. 8-21, Dec. 2017.

DERHAMY, H.; RÖNNHOLM, J.; DELSING, J.; ELIASSON, J.; DEVENTER, J. **Protocol interoperability of OPC UA in service-oriented architectures.** 15th International Conference on Industrial Informatics (INDIN), Emden, 2017, pp. 44-50.

DIVYA, K.; JEYALATHA, S. **Key technologies in cloud computing**. International Conference on Cloud Computing Technologies, Applications and Management (ICCCTAM), Dubai, 2012, pp.196-199, 8-10 Dec.

DUSTDAR, S.; PAPAZOGLOU, M. P. **Services and Service Composition** –An Introduction. Information Technology, v. 50, n. 2, p. 86–92, fev. 2008.

ERL, T. **SOA: Princípios de design de serviços**. Pearson Prentice Hall, São Paulo, v. 200, p. 18-21, 2009.

ESP8266. (2017). Wikipédia, a enciclopédia livre. Disponível em: https://pt.wikipedia.org/wiki/ESP8266. Acesso em: Janeiro, 2017.

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.

FATTORI, C. C.; JUNQUEIRA, F.; SANTOS FILHO, D. J. DOS; MIYAGI, P. E. **Service composition modeling using interpreted Petri net for system integration**. In: IEEE International Conference on Mechatronics, 2011, pp. 696-701.

FELDHORST, S.; LIBERT, S.; HOMPEL, M.; KRUMM, H. **Integration of a legacy automation system into a SOA for Devices**. In: Proceedings of IEEE Conference on Emerging Technologies & Factory Automation, 2009. ETFA 2009, 2009, pp. 1–8.

FERREIRA, I. V.; BIGHETI, J. A.; MANSANO, R. K.; GODOY, E. P. **Proposta de um Modelo Para a Aplicação da Internet das Coisas Industrial**. In: XIII Simpósio Brasileiro de Automação Inteligente (XIII SBAI), 2017, Porto Alegre. Anais do XII SBAI. Porto Alegre: SBA, 2017.

FISCHIONE, C.; PARK, P.; DI MARCO, P.; JOHANSSON, K.H. **Design Principles of Wireless Sensor Networks Protocols for Control Applications**, S.K. Mazumder (ed.), Wireless Networking Based Control, Ch. 9, Springer, 2011, pp. 203-238.

FOWLER, M. **Microservices a definition of this new architectural term**, 2014. Disponível em: https://martinfowler.com/articles/microservices.html. Acesso em: fevereiro, 2018.

FRANCESCO, P.D.; LAGO, P.; MALAVOLTA, I. **Migrating Towards Microservice Architectures**: An Industrial Survey. IEEE International Conference on Software Architecture (ICSA), Seattle, Washington, USA, 2018, pp. 29-2909.

FRANCESCO, P.D.; MALAVOLTA, I.; LAGO, P. **Research on Architecting Microservices:** Trends, Focus, and Potential for Industrial Adoption. IEEE International Conference on Software Architecture (ICSA), Gothenburg, 2017, pp. 21-30.

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.

GORBACH, G.; MICK, R. **Collaborative manufacturing management strategies**. In: ARC Strategies, White Paper, November, 2002.

GUBBI, J; BUYYA, R.; MARUSIC, S.; PALANISWAMI, M. **Internet of Things (IoT): A vision, architectural elements, and future directions.** Future generation computer systems, v. 29, n. 7, p. 1645-1660, 2013.

GUPTA, R.A.; CHOW, M.Y. **Networked Control System:** Overview and Research Trends. IEEE Transactions on Industrial Electronics, Vol. 57, No. 7, pp. 2527-2535, July, 2010.

HE, H.; MAPLE, C.; WATSON, T.; TIWARI, A.; MEHNEN, J.; JIN, Y.; GABRYS, B. **The security challenges in the IoT enabled cyber physical systems and opportunities for**

evolutionary computing & other computational intelligence. 2016 IEEE Congress on Evolutionary Computation, CEC 2016, pp. 1015–1021, 2016.

HEGAZY, T.; HEFEEDA, M. **Industrial Automation as a Cloud Service**, IEEE Transactions on Parallel and Distributed Systems, Vol.26, No.10, pp.2750-2763, Oct. 1.

HENNEKE, D.; ELATTAR, M.; JASPERNEITE, J. **Communication patterns for Cyber-Physical Systems**, 2015. IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA), Luxembourg, 2015, pp. 1-4.

HERMANN, M.; PENTEK, T.; OTTO B. **Design Principles for Industrie 4.0 Scenarios:** A Literature Review. Working Paper No.01, 2016.

HUI, J. W.; CULLER, D. E. **IP is dead, long live IP for wireless sensor networks**. In: Proceedings of the 6th ACM conference on Embedded network sensor systems. ACM, 2008. p. 15-28.

HUTCHISON, T. **Service Oriented Architecture (SOA) Security**. Disponível em: https://dokumen.tips/documents/troy-hutchison.html. Acesso em: Janeiro, 2019.

IMC-AESOP Project. (2011). **IMC-AESOP Project**. Disponível em: http://www.imcaesop.eu. Acesso em: Novembro, 2018.

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.

INTERNET ENGINEERING TASK FORCE. (2017). **Constrained application protocol:** request for comments 7252. Disponível em: https://tools.ietf.org/html/rfc7252. Acesso em: Agosto, 2017.

IRSHAD M. **A Systematic Review of Information Security Frameworks in the Internet of Things (IoT)**. IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS). DOI: 10.1109/HPCC-SmartCity-DSS.2016.0180. 12-14 Dec. 2016

JAMMES, F.; KARNOUSKOS, S.; BONY, B.; NAPPEY, P.; COLOMBO, A.W.; DELSING, J.; ELIASSON, J.; KYUSAKOV, R.; STLUKA, P.; TILLY, M.; BANGEMANN, T. **Promising Technologies for SOA-based Industrial Automation Systems**. In: Bangemann T, Karnouskos S, Delsing J, Stluka P, Harrison R, Jammes F, Lastra JL (eds), Ch 4, Industrial cloud-based cyber-physical systems. Springer International Publishing, Switzerland, 2014, pp 01–22.

JAMMES, F.; SMIT, H. - **Service-oriented paradigms in industrial automation**. IEEE Transactions on Industrial Informatics, v. 1, n. 1, p. 62-70, 2005.

JSMODBUS. (2018). **Simple Modbus TCP Master/Slave implementation for Node.js**. Disponível em: http://github.com/dcvice1967/jsModbus. Acesso em: Janeiro, 2018.

JSON. (2017). **Introducing JSON**. Disponível em: http://www.json.org. Acesso em: Outubro, 2017.

KAGERMANN, H.; WAHLSTER, W.; HELBIG, J. **Recommendations for implementing the strategic initiative INDUSTRIE 4.0:** Final report of the Industrie 4.0 Working Group. [s.l: s.n.], 2013. Disponível em: https://www.din.de/blob/76902/e8cac883f42bf28536e7e8165993f1fd/recommendations-for-implementing-industry-4-0-data.pdf. Acesso em: Maio, 2019

KANG, H. S.; LEE, J. Y.; CHOI, S.; KIM, H.; PARK, J. H.; SON, J. Y.; KIM, B. H.; NOH, S. Do. **Smart manufacturing:** Past research, present findings, and future directions. International

Journal of Precision Engineering and Manufacturing - Green Technology, v. 3, n. 1, p. 111–128, 2016.

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.

KIM, J.; KIM, J.; LEE, J; YUN, J. **M2M service platforms:** Survey, issues, and enabling technologies. IEEE Communications Surveys and Tutorials, v. 16, n. 1, p. 61-76, 2014.

KOVATSCH, M. **Scalable Web Technology for the Internet of Things**. Tese de Doutorado. ETH Zurich, Zurich, 2015.

KRAMP, T.; VAN KRANENBURG, R.; LANGE, S. **Introduction to the Internet of Things. In: Enabling Things to Talk**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 1–10.

KULKARNI, A. **IoT and SCADA:** Complimentary technologies for Industry 4.0, 2016. Disponível em: http://altizon.com/iotandscadacomplimentarytechnologiesthataresteppingstonetoindus-try40/. Acesso em: 2016.

KUMAR, B. V.; NARAYAN, P.; NG, T. **Implementando SOA Usando Java EE**. Rio de Janeiro: Alta Books, 2012.

KUMAR, R.; BOSE, A. K. **Internet of Things and OPC UA**. The Eleventh International Conference on Networking and Services, n. c, p. 38–43, 2015.

KUSHIDA, T.; PINGALI, G.S. **Industry Cloud** - Effective Adoption of Cloud Computing for Industry Solutions. In IEEE 7th International Conference on Cloud Computing, 2014, pp.753-760.

LEE, J.; BAGHERI, B.; KAO, H. **A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems**. Manufacturing Letters. Vol. 3, 2015, pp. 18-23.

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

LEITNER, S.; MAHNKE, W. **OPC UA** - Service-oriented Architecture for Industrial Applications. Softwaretechnik-Trends, 2006.

LEWIS, J; FOWLER, M. **Microservices**. Martin Fowler, 2014. Disponível em: https://martinfowler.com/articles/microservices.html. Acesso em: 10 abr. 2015.

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

LYDON, B. **Internet of Things Industrial automation industry exploring and implementing IoT**. InTech Magazine, v. 2, 2014. Disponível em: https://www.isa.org/standards-and-publications/isa-publications/intech-magazine/2014/mar-apr/cover-story-internet-of-things/. Acesso em: Agosto, 2018.

MANYIKA, J. **The Internet of Things:** Mapping the value beyond the hype. McKinsey Global Institute, 2015.

MARTINS, I.R.; ZEM, J.L. **Estudo dos Protocolos de Comunicação MQTT e CoAP para Aplicações Machine-To-Machine e Internet das Coisas.** Revista Tecnológica da Fatec Americana, v. 3, n. 1, p. 24, 2016.

MARZULLO, F. P. **SOA na prática:** inovando seu negócio por meio de soluções orientadas a serviços. São Paulo: Novatec Editora, 2009.

MAYA. (2019). **The MAYA Project**. Disponível em: http://www.maya-euproject.com/. Acesso em: Janeiro, 2019

MELO, J. I. G.; SOUIT, S.; FILHO, D. J. S.; JUNQUEIRA, F.; MIYAGI, P. E. **A systematical approach to expose manufacturing system as a service**. 2010 9th IEEE/IAS International Conference on Industry Applications, INDUSCON 2010, 2010.

MENDES, J.M.; BEPPERLING, A.; PINTO, J.; LEITAO, P.; RESTIVO, F.; COLOMBO, A.W. **Software methodologies for the engineering of service-oriented industrial automation:** the continuum project. In: Proceedings of the 33rd Computer Software and Applications Conference (COMPSAC'09), 2009, 452–459.

MOLECULER. (2018). **Fast & powerful microservices framework for Node.js**. Disponível em: https://moleculer.services/. Acesso em: Setembro, 2018.

MOLECULER. (2019). **Moleculer Polyglot Implementations**. Disponível em: https://github.com/moleculerjs/awesome-moleculer#polyglot-implementations. Acesso em: Janeiro, 2019

NEWMAN, S. **Building Microservices**. O'Reilly Media, Inc. CA: Gravenstein, 2015.

NIKOLAIDIS, P.; DIDIC, A.; MUBEEN, S.; PEIBREIVOLD, H.; SANDSTROM, K. AND BEHNAM, M. (2015). **Applying Mitigation Mechanisms for Cloud-based Controllers in Industrial IoT Applications**. Malardalen Real-Time Research Centre (MRTC), Malardalen University, Sweden, ABB Corporate Research, 2015.

NIMBLE. **The NIMBLE Project**. Disponível em: https://www.nimble-project.org/. Acesso em: Janeiro, 2019

NODE-COAP. **A client and server library for CoAP modelled after the http module**. Disponível em: http://github.com/mcollina/node-coap. Acesso em: Janeiro, 2018.

NODEMCU DEVKIT. (2018). Disponível em: https://github.com/nodemcu/nodemcu-devkit-v1.0. Acesso em: Janeiro, 2018.

OPENPLC. (2017). Disponível: www.openplcproject.com. Acesso em: 2017.

PANIAGUA, C.; ELIASSON, J.; DELSING, J. **Efficient Device-to-Device Service Invocation Using Arrowhead Orchestration**. IEEE Internet of Things Journal, vol. 7, no. 1, pp. 429-439, Jan. 2020.

PAPAZOGLOU, MIKE P. **Service -Oriented Computing:** Concepts, Characteristics and Directions. Tilburg University INFOLAB, Dept. of Information Systems and Management, 2006.

PAUTASSO, C.; ZIMMERMANN, O.; LEYMANN, F. **Restful web services vs. big web services:** making the right architectural decision. In: Proceedings of the 17th International Conference on World Wide Web, 2008, pp. 805–814.

PISCHING, M. A. **Arquitetura para descoberta de equipamentos em processos de manufatura com foco na indústria 4.0**. Tese de Doutorado, Escola Politécnica, Engenharia de Controle e Automação Mecânica, Universidade de São Paulo (USP), São Paulo, 2017.

PRODUCTIVE 4.0. Productive 4.0 Project Website. Disponível em: https://productive40.eu/. Acesso em: Abril, 2019.

RAJKUMAR, R.; LEE, I.; SHA, L.; STANKOVIC, J. **Cyber-Physical Systems:** The Next Computing Revolution, Proceedings of the Design Automation Conference 2010, California, USA. pp. 731-736.

RICHARDSON, C.; SMITH, F. **Microservices From Design to Deployment**. NGINX, 2016. Disponível em: https://www.nginx.com/resources/library/designing-deploying-microservices/. Acesso em: Janeiro, 2019

SARKAR, S.; VASHI, G.; ABDULLA, P. **Towards Transforming an Industrial Automation System from Monolithic to Microservices**. IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA), Turin, 2018, pp. 1256-1259.

SAUTER, T.; SOUCEK, S.; KASTNER, W.; DIETRICH, D. **The Evolution of Factory and Building Automation**. IEEE Industrial Electronics Magazine, vol.5, no.3, pp.35-48, Sept, 2011.

SHELBY, Z. **Constrained RESTful environments (CoRE) link format**. 2012. Disponível em: https://tools.ietf.org/html/rfc6690. Acesso em: Janeiro, 2019

SHELBY, Z. HARTKE, K. BORMANN, C. **CoAP Specification**, 2014. Disponível em: https://datatracker.ietf.org/doc/rfc7252/. Acesso em: Janeiro, 2019

SHENG, Q. Z.; QIAO, X.; VASILAKOS, A. V.; SZABO, C.; BOURNE, S.; XU, X. **Web services composition:** A decade's overview. Information Sciences, v. 280, October 2014, pp. 218– 238.

SIRENA. (2006). **ITEA 02014 SIRENA Project**. Disponível em: https://itea3.org/project/sirena.html. Acesso em: Setembro, 2018

SISINNI, E., SAIFULLAH, A., HAN, S. **Industrial Internet of Things: Challenges, Opportunities, and Directions.** Transactions on Industrial Informatic. IEEE, 2018.

SOCRADES. (2008). **EU FP6 IST-5-034116 SOCRADES Project**. Disponível em: http://www.socrades.net. Acesso em: Novembro, 2018.

SODA. (2007). **ITEA 05022 SODA Project**, 2007. Disponível em: https://itea3.org/project/**soda**.html. Acesso em: Novembro, 2018.

SONG, J.; MOK, A.; CHEN, C.; NIXON, M.; BLEVINS, T.; WOJSZNIS, W. **Improving PID control with unreliable communications.** *Paper presented at the ISA EXPO Technical Conference.* Houston, Texas, October 17-19, 2006.

SOUIT, S. **Orquestração de sistemas produtivos dispersos**. Biblioteca Digital de Teses e Dissertações da Universidade de São Paulo, São Paulo, 2013. Disponível em: http://www.teses.usp.br/teses/disponiveis/3/3152/tde-26062014110055/. Acesso em: julho, 2017.

SOUIT, S.; FATTORI, C. C.; JUNQUEIRA, F.; FILHO, D. J. S.; MIYAGI, P. E. **Orchestrating dispersed productive systems**. In: IEEE International Conference on Emerging Technologies and Factory Automation, ETFA, Anais...2013.

STANKOVIC, J.A. **Research Directions for the Internet of Things**, IEEE Internet of Things Journal, Vol.1, No.1, 2014, pp.3-9.

SUNDMAEKER, H.; GUILLEMIN, P.; FRIESS, P. **Vision and challenges for realising the Internet of Things**. Cluster of European Research Projects on the Internet of Things, European Commision, v. 3, n. 3, p. 34-36, 2010.

TAJAMMUL, M.; PARVEEN, R. **Comparative Analysis of Big Ten ISMS Standards And Their Effect On Cloud Computing**. 2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN). IEEE Conferences things. Advanced Computer Theory and Engineering (ICACTE). 3rd International. Gurgaon, India, 12-14 Oct, 2017, pp. 362 – 367

THEORIN, A.; HAGSUND, J.; JOHNSSON, C. **Service orchestration with OPC UA in a graphical control language**, Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA), Barcelona, 2014, pp. 1-6.

UNGUREAN, I.; GAITAN, N. C.; GAITAN, V. G. **A Middleware Based Architecture for the Industrial Internet of Things**. KSII Transactions on Internet & Information Systems, v. 10, n. 7, 2016.

UniPi. Extension Board for Raspberry Pi Model 1.1. Disponível em: https://www.unipi.technology/unipi-1-1-p36. Acesso em: Setembro, 2019.

WANG, S.; WAN, J.; LI, D.; ZHANG, C. **Implementing Smart Factory of Industrie 4.0:** An Outlook. International Journal of Distributed Sensor Networks, v. 12, n. 1, p. 10, 2016.

WANG, Y.; TOWARA, T.; ANDERL, R. **Topological Approach for Mapping Technologies in Reference Architectural Model Industrie 4.0 (RAMI 4.0)**. Proceedings of the World Congress on Engineering and Computer Science 2017, Vol II, WCECS, 2017.

WEYER, S., SCHMITT, M., OHMER, M.; GORECKY, D. **Towards Industry 4.0-Standardization as the crucial challenge for highly modular, multi-vendor production systems**. IFAC-Papers on Line, 48(3), 2015, pp. 579- 584.

WILL, L.; KÖPPEN, V.; SAAKE G., **Flexibility in SOA Operations:** The Need for a Central Service Component. IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations, 2014, pp. 306-315.

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, vol.11, 2017, pp.17-27.

WU D., GREER M., ROSEN D.; SCHAEFER D. **Cloud manufacturing:** Strategic vision and state-of-the-art, J. Manuf. Syst., Vol. 32, No. 4, 2013, pp. 564–579.

XIAO, Z.; WIJEGUNARATNE, I.; QIANG, X. **Reflections on SOA and Microservices**. In: 4th IEEE International Conference on Enterprise Systems, 2016, pp. 60-67.

XU, L.; HE, W.; LI, S. **Internet of Things in Industries:** A Survey. IEEE Transactions on Industrial Informatics, v. 10, n. 4, 2014, pp. 2233–2243.

XU, X. **From cloud computing to cloud manufacturing**. Robotics and Computer Integrated Manufacturing, v. 28, n. 1, 2012, pp. 75–86.

ZHONG C.; ZHU Z.; HUANG R., **Study on the IOT Architecture and Access Technology**. 16th International Symposium on Distributed Computing and Applications to Business, Engineering and Science, IEEE, 2017.

ZHOU, J.; CAO, Z.; DONG, X.; VASILAKOS, A.V. **Security and Privacy for Cloud-Based IoT:** Challenges. IEEE Communications Magazine, vol. 55, no. 1, 2017, pp. 26–33.

ZUEHLKE, D. **Smart Factory** — Towards a factory-of-things, Annual Reviews in Control, Vol. 34, No. 1, 2010, pp. 129-138.

10 APÊNDICE

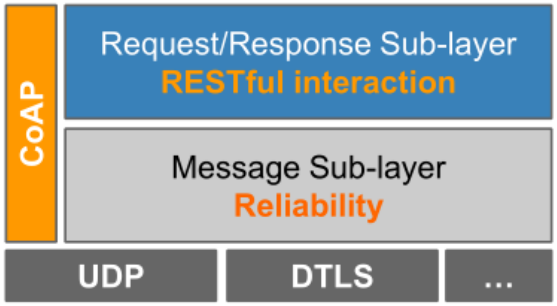
10.1 PROTOCOLO COAP

Um grupo de trabalho da IETF (*Internet Engineering Task Force*) chamado CoRE (*Constrained RESTful Environments*) criou o CoAP (*Constrained Application Protocol*). As atividades do grupo começaram em março de 2010 com o objetivo de criar uma estrutura para aplicativos que manipulam recursos simples incorporados em dispositivos interconectados em redes limitadas. O CoAP é um protocolo de aplicação destinado a ser usado em dispositivos eletrônicos, permitindo a comunicação interativa com a Internet e outros dispositivos, sendo a RFC 7252 a especificação do protocolo.

O *Constrained Application Protocol* (CoAP) é um protocolo de comunicação desenvolvido para dispositivos com baixo consumo energético, processamento e em redes limitadas. Com frequência esses dispositivos têm microcontroladores de 8 bits, com quantidades reduzidas de memória e estão se comunicando através de redes com alta taxa de perda de pacotes e baixa velocidade. É um protocolo pensado para aplicações M2M (SHELBY; HARTKE; BORMANN, 2014).

O objetivo do CoAP não é apenas comprimir o HTTP, mas sim ter um subconjunto de características REST em comum com o HTTP, mas otimizadas para aplicações M2M (SHELBY; HARTKE; BORMANN, 2014). CoAP é baseado na troca de mensagens sobre UDP/IP, que oferece um melhor desempenho do que o TCP/IP para comunicação sem fio de baixo consumo de energia e em redes com altas taxas de perda de pacotes (HUI & CULLER, 2008). Em alguns casos, existe a necessidade de mais garantias de entrega das mensagens do que o UDP pode fornecer, sendo implementada para isso uma fina camada de controle, além da camada de requisição-resposta, conforme a Figura 92. O CoAP pode ser dividido em duas subcamadas: A subcamada de requisição/resposta, responsável pelas interações RESTful e a subcamada de mensagem, que gerencia o reenvio de mensagens.

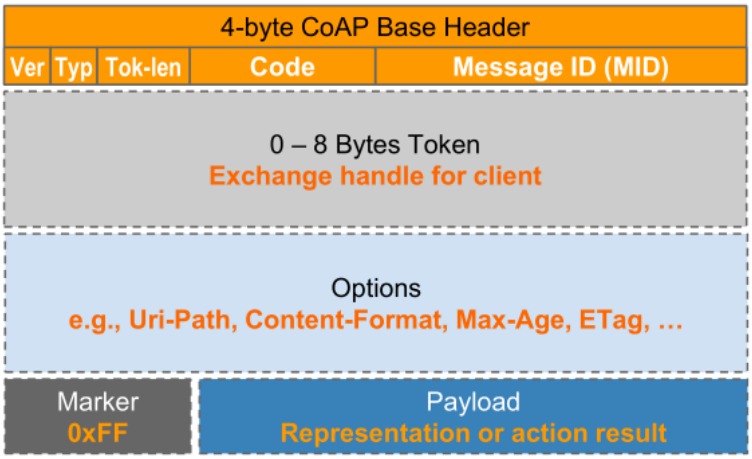
Figura 92 - Divisão em Camadas do Protocolo CoAP



Fonte: Kovatsch (2015).

As mensagens são codificadas em um formato binário com um cabeçalho de 4 bytes, seguido por um token de largura variável (de 0 a 8 bytes). Uma série de opções do CoAP no formato *Type-Length-Value* (TLV) pode vir ou não depois do token, que são seguidas também opcionalmente por um payload, ocupando o resto do datagrama. A Figura 93 mostra o formato da mensagem CoAP (MARTINS; ZEM, 2015).

Figura 93 - Frame CoAP



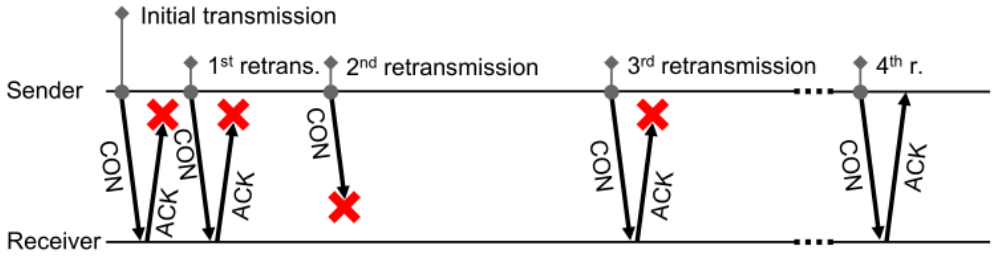
Fonte: Kovatsch (2015)

A subcamada de mensagem usa um identificador chamado *Message Identifier* (MID) para detectar mensagens duplicadas e quatro tipos de mensagem para aumentar a confiabilidade de entrega. O MID é compacto, tem 16 bits e permite a troca de cerca de 250 mensagens por segundo entre dois dispositivos. A confiabilidade de entrega da mensagem pode ser aumentada, marcando a mensagem como *Confirmable* (CON). A mensagem *Confirmable* é retransmitida em determinados intervalos de tempo, até que o destinatário envie uma mensagem de *Acknowledgement* (ACK) com o mesmo MID de mensagem, conforme Figura 94.

Ao receber uma mensagem CON, o receptor precisa responder com uma confirmação de recebimento (ACK). Caso uma das mensagens (CON ou ACK) seja perdida, a mensagem CON é reenviada até quatro vezes, com um intervalo inicial aleatório de 2 a 3 segundos entre cada mensagem, que é dobrado após cada retransmissão. Se o destinatário não for capaz de processar a mensagem, ele responde com uma mensagem *Reset* (RST) ao invés do *Acknowledgement* (ACK). Uma mensagem que não precise de tanta confiabilidade na transmissão pode ser enviada como uma mensagem *Non-confirmable* (NON). Nesses casos, não há *Acknowledgement* por parte do destinatário, mas a mensagem ainda tem um MID para detecção de duplicatas. Se o receptor não for capaz de processar a mensagem, ele pode responder com

uma mensagem Reset (RST) (SHELBY, HARTKE & BORMANN, 2014; KOVATSCH, 2015).

Figura 94 - Procedimento de Comunicação via Mensagem com Confiabilidade de Entrega (CON) no CoAP



Fonte: Kovatsch (2015)

O token é gerado pelo cliente e espelhado pelo servidor, sendo necessário que seja único para cada requisição aberta ou, pelo menos, para cada destinatário, se o endereço do servidor também for utilizado para identificação das mensagens. Ele também pode ter tamanho de zero bytes, sendo chamado de empty token nesse caso, útil para minimizar o tamanho das mensagens. Convém notar que o que identifica uma resposta a uma requisição é a utilização do mesmo token e não o MID. Em alguns casos onde a resposta leva algum tempo, pode ser enviada uma mensagem ACK vazia, com o mesmo MID da requisição, enquanto a resposta é enviada posteriormente como uma mensagem CON com o mesmo token da requisição e um MID diferente. Mensagens NON sempre geram uma mensagem com MID diferente como resposta, precisando ser feita a correlação entre as mensagens através do token. Quando o cliente recebe uma resposta a uma mensagem CON que ainda não recebeu um ACK, ele interrompe as retransmissões, pois a resposta funciona como um *Acknowledgement* implícito. Uma mensagem CON vazia pode ser enviada para desencadear uma mensagem RST, comportamento que pode ser utilizado de modo similar ao PING, para testar a conectividade em uma rede (KOVATSCH, 2015).

Com funcionamento similar aos *headers* no HTTP, as requisições e respostas podem ter opções para especificar informações adicionais. Cada opção tem um número que a identifica. Uma mensagem pode conter várias opções, ordenadas em ordem crescente de acordo com seu identificador numérico. O identificador numérico não é utilizado diretamente, sendo utilizado um valor denominado delta, resultante de uma subtração entre o valor da opção que se deseja referenciar e da opção anterior. Caso não haja uma opção anterior, é considerado o próprio identificador numérico da opção como o delta. Além do delta, cada referência a uma opção presente na mensagem deve especificar o tamanho do valor da opção e seu valor propriamente

dito. O valor da opção pode ser uma sequência de bytes, que pode ter comprimento zero (campo valor vazio), um número inteiro não negativo ou uma string.

A subcamada de requisição/resposta implementa o modelo REST, o que faz com que o CoAP tenha muitas similaridades com o HTTP. Porém, como é voltado para dispositivos mais simples e para comunicação M2M, o CoAP tem algumas diferenças em relação ao HTTP, como o campo de código (Code -Figura 93), usado no CoAP para identificar os métodos, além do status das operações como acontece no HTTP. No CoAP são utilizados quatro métodos para interação com os recursos: GET, PUT, POST e DELETE. Eles apresentam mesma semântica e funcionamento dos métodos análogos do HTTP. Os códigos de status também apresentam algumas diferenças, como um ponto separando a classe do código do restante, além de alguns códigos mais explícitos, como o "2.02 *Deleted*" e o "2.04 *Changed*", que equivalem ao código "204 *No Content*" do HTTP. As mensagens de erro podem conter um payload com uma mensagem curta para que desenvolvedores entendam o motivo de uma falha (KOVATSCH, 2015).

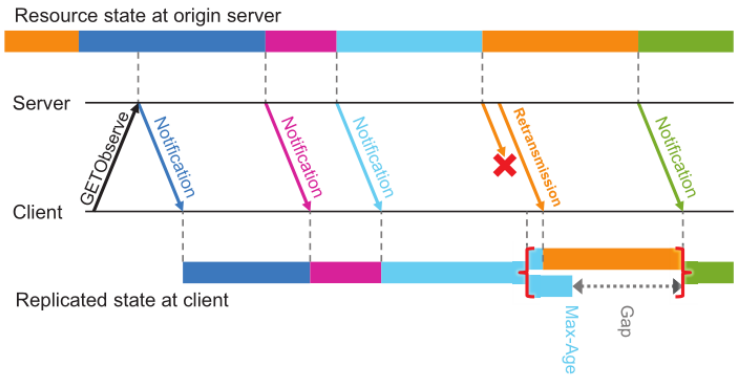
Existem extensões que adicionam funcionalidades ao protocolo CoAP. Uma delas é a funcionalidade de observar recursos. A implementação dessa funcionalidade é feita sobre uma mensagem com método GET, onde o cliente adiciona a opção "*observe*" igual a zero. Isso indica ao servidor que deve incluir o cliente na lista de "*observers*". Caso o servidor não consiga incluir o cliente na lista, ou não suporte essa funcionalidade, a resposta torna-se uma resposta padrão a uma requisição GET e não contém a opção "*observe*".

Ao incluir o cliente na lista, o servidor passa a enviar respostas ao cliente conforme o recurso observado se altera, fazendo o possível para manter a sincronia entre a representação do recurso no servidor e no cliente. Essas respostas são chamadas de notificações e são relacionadas com a requisição original através do token. Isso torna o modelo requisição/resposta em um modelo requisição/múltiplas-respostas. Convém notar que, diferentemente de um modelo *publish/subscribe*, o servidor pode não enviar notificações todas as vezes em que houver alterações no recurso observado. Isso pode ocorrer por congestionamento na rede ou recursos se alterando muito rapidamente. As notificações têm uma opção "*Max-Age*", utilizada pelo servidor para indicar até quando o cliente pode considerar aquela notificação uma representação válida do recurso observado. Elas podem ficar armazenadas em cache para serem utilizadas caso alguma notificação se perca.

Caso a notificação se torne obsoleta, o cliente deve assumir que não está mais na lista de observadores e reenviar a requisição GET original. Isso pode gerar intervalos de tempo em que o cliente não tem uma representação válida daquele recurso (Figura 95) até receber uma resposta. Se o cliente deseja parar de observar um determinado recurso, basta enviar uma

requisição GET com a opção "*observe*" igual a um. Nas notificações, a opção "observe" apresenta um valor crescente, tornando possível ordená-las. Esse valor pode se iniciar em qualquer número e aumentar em passos arbitrários.

Figura 95 - Funcionamento da função Observe no CoAP

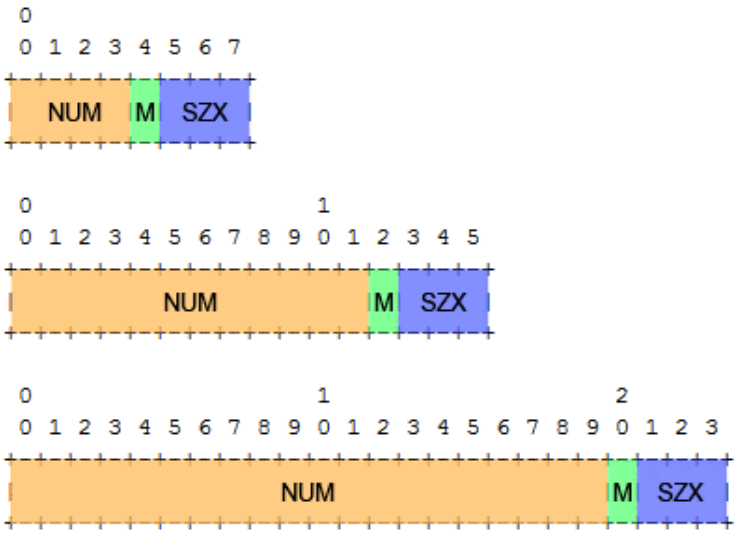


Fonte: Kovatsch (2015).

Outra extensão ao CoAP é a transferência block-wise, que é utilizada para dividir uma mensagem grande em várias mensagens menores. Isso é necessário, pois o CoAP é baseado em UDP/IP, que oferece apenas fragmentação de mensagens, o que pode ser problemático em um ambiente com restrições e redes com altas taxas de perdas de pacotes. A divisão das mensagens é feita através das opções "Block1" e "Size1", referentes à representação do recurso transferido na requisição e "Block2" e "Size2", referentes à representação do recurso transferido na resposta. O valor da opção Block é de tamanho variável, podendo ter entre 0 e 3 bytes. Conforme mostrado na Figura 96, ela deve conter o tamanho do bloco (SZX), quantos blocos virão a seguir (M) e o número relativo daquele bloco (NUM), em relação ao total de blocos. A opção Size indica o tamanho total estimado da representação do recurso que será transferida (BORMANN & SHELBY, 2016).

A descoberta de recursos contidos em um dispositivo também é feita por uma extensão do protocolo CoAP. O *Constrained RESTful Environments* (CoRE) *Link Format* padroniza essa descoberta de recursos voltada para a comunicação M2M. A URI "/.well-known/core" é definida como o ponto de entrada padrão para requisitar uma lista de links de recursos hospedados pelo servidor, possibilitando assim a descoberta de recursos. Esse mecanismo pode ser utilizado em outros protocolos web, como o HTTP (SHELBY, 2012).

Figura 96 - Formato da opção Block.



Fonte: Bormann & Shelby (2016).

A descoberta pode ser feita tanto por *unicast*, quanto por *multicast*, através de uma requisição com o método GET para a URI "/.well-known/core" relativa ao IP do servidor ou ao endereço de *multicast*, sendo encontrado no *payload* da resposta o *CoRE Link Format*. O *CoRE Link Format* é composto por links entre parênteses angulares e atributos que são conectados por ponto e vírgula, sendo utilizada a vírgula para separação entre os links.

A Figura 97 mostra um exemplo de utilização do *CoRE Link Format*, sendo possível ver os diferentes atributos utilizados. O atributo "*title*" contém um nome amigável para o recurso que pode ser lido por um usuário humano, enquanto "rt" indica uma etiqueta para classificar o recurso que deve ser previamente conhecida. Caso contrário, o cliente deve ser capaz de acessar o link indicado pelo atributo "if". O tamanho máximo da representação de um recurso pode ser indicado pelo atributo "sz", enquanto uma lista com os formatos possíveis dessa representação é indicada por "ct". Os atributos "anchor" e "rel" são utilizados para descrever uma relação entre dois recursos (KOVATSCH, 2015).

Figura 97 - Exemplo de utilização do CoRE Link Format. A quebra de linha após as vírgulas foi utilizada apenas para melhorar a legibilidade, não sendo utilizada na prática.

```
</sensors/temp>;rt="ucum:cel",
</large>;sz=1280;title="Large resource",
</multi-format>;ct="0 41";title="text/plain and application/xml",
</t>;anchor="/sensors/temp";rel="alternate";title="Shortcut to sensor",
</semantic>;rt="restdesc";if="http://api.example.com/usage/"
```

Fonte: Kovatsch (2015).