Dissertação de Mestrado

Inatel
Instituto Nacional de Telecomunicações

NovaGenesis: Aplicação de uma Arquitetura de Internet do Futuro para Sistemas Ciber-Físicos na Indústria 4.0

DIEGO GABRIEL SOARES PIVOTO

DEZEMBRO / 2021

NOVAGENESIS: APLICAÇÃO DE UMA ARQUITETURA DE INTERNET DO FUTURO PARA SISTEMAS CIBER-FÍSICOS NA INDÚSTRIA 4.0

DIEGO GABRIEL SOARES PIVOTO

Dissertação apresentada ao Instituto Nacional de Telecomunicações, como parte dos requisitos para obtenção do Título de Mestre em Engenharia de Telecomunicações.

ORIENTADOR: Prof. Dr. Antônio Marcos Alberti.

Santa Rita do Sapucaí
2021

Pivoto, Diego Gabriel Soares
P693n
NovaGenesis: Aplicação de uma Arquitetura de Internet do Futuro para Sistemas Ciber-Físicos na Indústria 4.0. / Diego Gabriel Soares Pivoto. – Santa Rita do Sapucaí, 2021.
119 p.

Orientador: Prof. Dr. Antônio Marcos Alberti.
Dissertação de Mestrado em Telecomunicações – Instituto Nacional de Telecomunicações – INATEL.
Inclui bibliografia e anexo.

1. Sistema Ciber-Físico Industrial 2. Indústria 4.0 3. Internet das Coisas Industrial 4. Internet do Futuro. 5. NovaGenesis. 6. Mestrado em Telecomunicações. I. Alberti, Antônio Marcos. II. Instituto Nacional de Telecomunicações – INATEL. III. Título.

CDU 621.39

Ficha Catalográfica elaborada pelo Bibliotecário da Instituição
CRB6-2718 – Paulo Eduardo de Almeida

Inatel
Mestrado em Telecomunicações

FOLHA DE APROVAÇÃO

Dissertação defendida e aprovada em _____ /_____ /_____,
pela comissão julgadora:

Prof. Dr. Antônio Marcos Alberti
INATEL

Prof. Dr. Yvo Marcelo Chiaradia Masselli
INATEL

Prof. Dr. Marcelo Eduardo Pellenz
PUC PR

Coordenador do Curso de Mestrado
Prof. Dr. José Marcos Câmara Brito

Agradecimentos

Inicialmente gostaria de agradecer a Deus pela oportunidade oferecida.

Agradeço à minha família por todo o apoio e motivação durante esta caminhada.

Agradeço ao meu orientador Prof. Antônio Marcos Alberti por sua dedicação, orientação, profissionalismo e incentivo.

Agradeço aos meus amigos e colegas do Laboratório de Tecnologias da Informação e Comunicações (ICT Lab) pela convivência, apoio e cooperação.

Agradeço ao Instituto Nacional de Telecomunicações (Inatel) e à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio e suporte durante esta caminhada, bem como a todos os professores por seus preciosos ensinamentos.

Por fim agradeço aos demais membros da banca, os senhores Yvo Marcelo Chiaradia Masselli e Marcelo Eduardo Pellenz, por sua disposição e valiosas contribuições.

Sumário

Lista de Figuras

Lista de Tabelas

Acrônimo

AS Administration Shell

BF Binding Forwarding

BT Bluetooth

AMPQ Advanced Message Queuing Protocol

C2NET Cloud Collaborative Manufacturing Networks

CAN Controller Area Network

CC Cloud Computing

CNC Controle Numérico Computadorizado

CoAP Constrained Application Protocol

CPMC Cyber-Physical Manufacturing Cloud

CPS Cyber-Physical System

DDS Data Distribution Service

DDSI-RTPS Data Distribution Real-Time Publish-Subscribe

DFC Data Collection Framework

DHCP Dynamic Host Configuration Protocol

D/IRAM Data/Instruction Random Access Memory

DRAM Data Random Access Memory

DTIM Delivery Traffic Indication Message

EMI Eletromagnetic Interference

FI Future Internet

FMI Functional Mockup Interface

GE General Electric

GIRS Generic Indirection Resolution Service

GSM Global System for Mobile

GW Gateway

HT Hash Table

HTS Hash Table Service

HTTP Hypertext Transfer Protocol

HMI Human-Machine Interface

I4.0 Indústria 4.0

I4.0C Industry 4.0 Component

IaaS Infrastructure as a Service

ICN Information-Centric Networking

ICPS Industrial Cyber-Physical System

ID Identificador

IIC Industrial Internet Consortium

IICF Industrial Internet Connectivity Framework

IIoT Industrial Internet of Things

IIRA Industrial Internet Reference Architecture

IoS Internet of Services

IoT Internet of Things

IP Internet Protocol

IPSec Internet Protocol Security

IRAM Instruction Random Access Memory

ISO International Organization for Standardization

LAN Local Area Network

LOC Localizador

LTE Long Term Evolution

M2M Machine-to-Machine

MQTT Message Queue Telemetry Transport

mA mili-Ampéres

ms milissegundos

NB Name Binding

NFV Network Function Virtualization

NG NovaGenesis

NLN Natural Language Names

NVS Non-Volatile Storage

OMG Object Management Group

OPC UA Open Platform Communications Unified Architecture

OS Operating System

OSI Open System Interconnection

PaaS Platform as a Service

PAN Personal Area Network

PFS Production Flow Schema

PG Proxy/Gateway

PGCS Proxy/Gateway/Controller Service

PGS Proxy/Gateway Service

PLC Programmable Logic Controller

PMML Predictive Model Markup Language

PN Petri Network

PS Publish/Subscribe

PSS Publish/Subscribe Service

QoS Quality-of-Service

RAM Random Access Memory

RAMI 4.0 Reference Architecture Model Industrie 4.0

RDF Resource Description Framework

REST Representative State Transfer

RESTful Web Representative State Transfer

RFID Radio-Frequency Identification

ROM Read-Only Memory

RTC Real-Time Clock

SaaS SOftware as a Service

SAS Superior System Administration Shell

SCN Service-Centric Networking

SCN Self-Certifying Name

SDK Software Development Kit

SDN Software-Defined Networking

SLA Service Level Agreement

SQL Standard Query Language

SSID Service Set Identifier

SOA Service-Oriented Architecture

SVN Self-Verifying Name

TCP Transmission Control Protocol

TIC Tecnologia de Informação e Comunicação

TI Tecnologia da Informação

TSN Time-Sensitive Networking

uDiT universal Digital Twin

UDP User Datagram Protocol

ULP Ultra Low Power

URL Uniform Resource Locator

XaaS Everything as a Service

XML Extensible Markup Language

Publicações

1. Diego G.S. Pivoto, Luiz F.F. de Almeida, Rodrigo da Rosa Righi, Joel J.P.C. Rodrigues, Alexandre Baratella Lugli, Antonio M. Alberti, Cyber-physical systems architectures for industrial internet of things applications in Industry 4.0: A literature review, Journal of Manufacturing Systems, Volume 58, Part A, 2021, Pages 176-192, ISSN 0278-6125, https://doi.org/10.1016/j.jmsy.2020.11.017.

Resumo

O cenário industrial vem passando por grandes mudanças devido ao aumento exponencial na demanda de produção e à iniciativa de Indústria 4.0 (I4.0), propondo a digitalização das fábricas. Dentre as diversas tecnologias emergentes, destacam-se *Industrial Internet of Things* (IIoT) e *Cyber-Physical System* (CPS), tornando o uso da Internet em ambientes fabris uma marca dessa nova era industrial. Nesse contexto, pesquisadores estão desenvolvendo sistemas IIoT e CPSs capazes de virtualizar ativos de rede e integrá-los a outros setores fabris e indústrias, garantindo a interoperabilidade entre diferentes arquiteturas. Devido às limitações da Internet atual e à necessidade de arquiteturas industriais avançadas para o suporte de tecnologias legadas e emergentes, este trabalho tem como objetivo o uso da arquitetura de *Future Internet* (FI) NovaGenesis (NG) para o desenvolvimento de um CPS em I4.0 baseado no *Reference Architecture Model Industry* (RAMI) 4.0. A metodologia de avaliação empregada foi baseada na realização de experimentos laboratoriais com equipamentos fornecidos pelo Instituto Nacional de Telecomunicações (Inatel) e na elaboração de testes relacionados à digitalização e ao monitoramento remoto de dados de dispositivos industriais em tempo real. Os resultados dos testes mostram a potencial viabilidade da utilização da arquitetura NG para desenvolver um *Industrial Cyber-Physical System* (ICPS) atendendo aos requisitos do modelo RAMI 4.0, combinando iniciativas de I4.0 e FI para desenvolver um meio eficiente de digitalização de dispositivos em ambiente industrial.

Pivoto, D. G. S. [dissertação de mestrado]. Santa Rita do Sapucaí: Instituto Nacional de Telecomunicações; 2021.

Palavras-chave: Sistema Ciber-Físico Industrial; Indústria 4.0; Internet das Coisas Industrial; Internet do Futuro; NovaGenesis.

Abstract

The industrial scenario has been undergoing exponential changes due to the increasing production demand and the Industry 4.0 (I4.0) initiative, which proposes the factory digitization. Among the various emerging technologies, it can be can highlighted the Industrial Internet of Things (IIoT) and the Cyber-Physical System (CPS), making the use of Internet in industrial environments a mark of this new industrial era. In this context, researchers are developing IIoT systems and CPSs capable of virtualizing network assets and integrating them with other manufacturing sectors and industries, ensuring the interoperability among different architectures. Due to the current Internet limitations and need for advanced architectures for supporting emerging and legacy technologies, this work aims to use a Future Internet (FI) architecture called NovaGenesis (NG) for the development of a CPS in I4.0 based on the Reference Architecture Model Industry (RAMI) 4.0. The evaluation methodology used was based on the performance of laboratory experiments with equipment used in the field provided by Intituto Nacional de Telecomunicações (Inatel) and the development of tests related to digitization and data remote monitoring of industrial devices in real-time. The results of the tests show the potential feasibility by using the NovaGenesis architecture to develop an Industrial Cyber-Physical System (ICPS) meeting the needs of the RAMI 4.0 model, combining I4.0 and FI initiatives to develop an efficient means of digitizing devices in the industrial environment.

Pivoto, D. G. S. [masters dissertation]. Santa Rita do Sapucaí: Instituto Nacional de Telecomunicações; 2021.

Keywords: Industrial Cyber-Physical System; Industry 4.0; Industrial Internet of Things; Modbus TCP/IP; ESP32; Future Internet; NovaGenesis.

Capítulo 1

Introdução

O impacto do aumento exponencial da demanda de produção industrial tem proporcionado às indústrias a necessidade de mudanças no processo de manufatura para acompanhar o ritmo de desenvolvimento tecnológico fabril. Essa crescente necessidade de aumento da produção, eficiência e qualidade produtiva levou a comunidade industrial a desenvolver conjuntamente novas tecnologias capazes de atender as necessidades e exigências de seus clientes [1]. A Figura 1.1 ilustra a evolução da produção industrial.

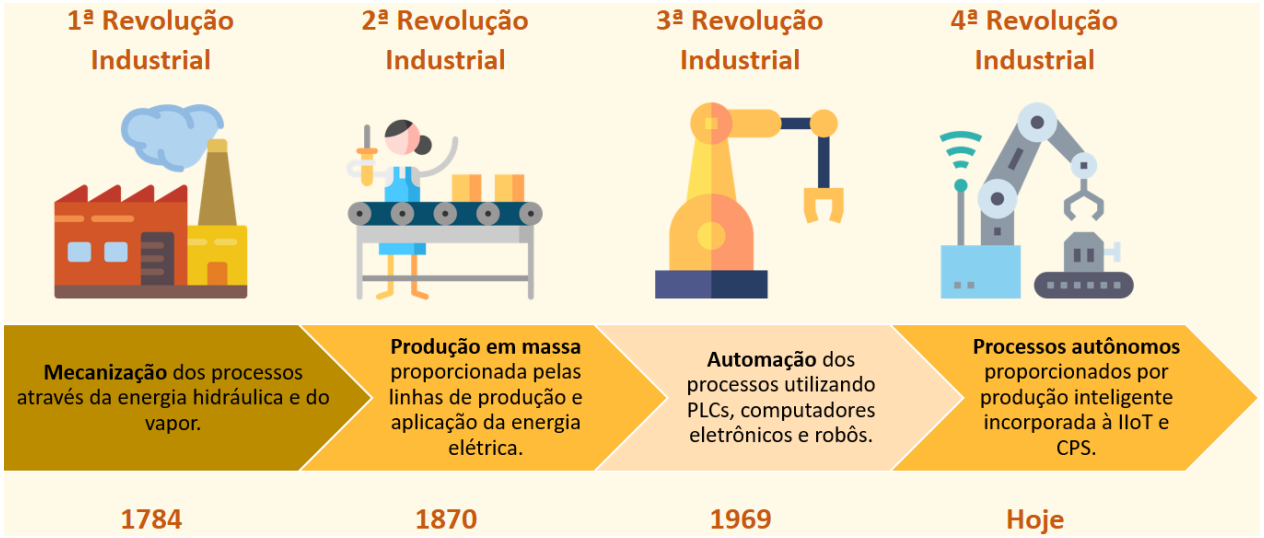

Figura 1.1: Evolução da produção industrial.

Primeiramente, em meados do século XVIII e XIX, a Primeira Revolução Industrial surgiu com a mecanização da produção de água e energia à vapor. Já no século XX, a Segunda Revolução Industrial foi marcada pela introdução da eletricidade nas fábricas e do processo de produção em massa. Além disso, houve a Terceira Revolução Industrial com o surgimento

das Máquinas de Controle Numérico Computadorizado (CNC), robôs industriais e tecnologias de informação e automação [2], [3].

No século XXI, estamos vivenciando a Quarta Revolução Industrial, também conhecida como Indústria 4.0 [4], [5], [6], que por sua vez é uma iniciativa alemã para automatizar o sistema de produção de forma eficiente. Dentre as tecnologias emergentes nessa nova era, podem ser destacados ICPSs [7], [8], [9] e IIoT [10], [11] para o desenvolvimento das chamadas *Smart Factories*, em português, Fábricas Inteligentes, cuja proposta principal é a digitalização das indústrias, através da virtualização, identificação, localização, gerenciamento e controle de todos os seus bens tangíveis capazes de gerar valor agregado, também conhecidos como ativos industriais.

ICPSs são sistemas automatizados capazes de realizar a integração do mundo físico com o mundo virtual, através do processo de digitalização dos ativos via uma rede de comunicação e infraestrutura de computadores [12]. Isso torna possível o gerenciamento e controle de todo o processo de manufatura de forma remota, proporcionando um ambiente industrial interativo. Já a IIoT possui uma proposta semelhante aos ICPSs, com o foco no desenvolvimento de uma rede de Internet industrial de dispositivos com identificadores únicos e localizadores, capaz de realizar virtualmente a coleta, monitoramento, controle e troca de seus dados [13].

Para a implementação de ICPSs e de uma rede IIoT em uma fábrica, fatores críticos devem ser levados em consideração, como por exemplo a interferência eletromagnética, também conhecida como *Eletromagnetic Interference* (EMI), provocada por ruídos de motores e outras máquinas [14]. Consequentemente, isso pode afetar o tempo de resposta dos ativos quando submetidos à ações de leitura/escrita de forma remota, visto que poucos milissegundos podem ser extremamente impactantes em um processo industrial, podendo ocasionar falhas e erros nos setores fabris.

Dentre os modelos de referência de arquitetura ICPS existentes, o *Reference Architectural Model Industrie* 4.0 (RAMI 4.0) se destaca pela sua proposta de *Service Oriented Architecture* (SOA) capaz de descrever os níveis hierárquicos de um sistema de manufatura em rede via Internet, o ciclo de vida de seus sistemas e produtos, e principalmente pela estrutura de Tecnologia da Informação (TI) de um Componente da Indústria 4.0, também conhecido como I4.0 *Component* (I4.0C), o que torna os ativos industriais com identificadores únicos e os integra com o mundo virtual [15]. Desta forma, propostas tem sido desenvolvidas afim de atender aos requisitos deste modelo de referência.

Atualmente, existem também discussões relacionadas à Indústria 5.0, tendo como definição a combinação das tecnologias de Indústria 4.0 com foco na aproximação do homem à fábrica, tendo como principais características:

- Conciliação e maior colaboração e iteração entre homem e máquina;
- Automação industrial e capital cognitivo;
- Necessidade de equilíbrio entre trabalho humano e maquinário;
- Tecnologias de I4.0 com exploração do lado criativo humano.

A utilização de ICPSs no ambiente industrial e consequentemente o aumento exponencial de dispositivos na rede devido à IIoT resulta em um grande impacto das limitações da Internet atual na indústria, como por exemplo: o endereçamento *Internet Protocol* (IP), vista a limitação física da quantidade de dispositivos que podem ser endereçados desta forma; a estrutura de identificação e localização dos elementos da rede (endereço IP como identificador e localizador), visto que a mudança de localização altera o identificador, que era para ser único em dispositivos industriais; e a heterogeneidade da rede (roteamento IP como mecanismo único de entrega), sendo ineficaz em um ambiente mais heterogêneo com redes de características diversas, impactando em uma rede industrial inflexível e com dificuldade na adaptação de novas tecnologias associadas à mesma.

Assim sendo, para se desenvolver um ICPS ideal, deve-se levar em consideração o modelo de arquitetura da Internet atual, suas limitações e questões de segurança [16], [17]. Visto que a evolução tecnológica é uma realidade além do ambiente fabril, pesquisadores de todo o mundo buscam criar uma rede mais segura e eficiente, desenvolvendo alternativas de arquiteturas FI [18], [19], com abordagens evolutivas e/ou revolucionárias de arquiteturas de rede semelhantes à Internet capazes de solucionar ou diminuir o impacto das limitações da Internet atual na indústria.

Dentre essas alternativas, pode ser destacada a NG, cujo modelo é uma arquitetura SOA definida por *software* com capacidade de auto-organização e compatibilidade com dispositivos móveis [20]. A NG adota o paradigma de Internet de Informação e Serviços, popularmente conhecido como *Internet of Information and Services* (IoIS), onde abordagens centradas a nomes, serviços e informações são combinados para desenvolver o tratamento, armazenamento e troca de informações na rede. Desta forma, a NG é uma arquitetura promissora capaz de atender ao modelo RAMI 4.0 através de suas características gerais, tendo como diferencial uma característica única e adicional de arquitetura direcionada à nomeação que, além de representar unicamente ativos industriais, realiza uma integração com as intenções baseada em linguagem, permitindo que as pessoas se expressem usando linguagem natural e consequentemente aproximando o homem do processo fabril, sendo estes ideais alinhados para um direcionamento inicial de uma arquitetura modelo para o futuro industrial, que é a Indústria 5.0.

1.1 Objetivos

Baseado nas afirmações anteriores, esta dissertação tem como objetivo avaliar o uso da NG como uma arquitetura capaz de atender aos requisitos do modelo de referência ICPS do RAMI 4.0 com características inovadoras que auxiliam nos problemas relacionados às limitações da Internet atual descritos acima. Esta avaliação será feita através de testes de monitoramento de dispositivos industriais usando a arquitetura centrada a nomes para identificação única dos ativos, de modo a verificar se a mesma atende aos requisitos do modelo de referência ICPS. Sendo assim, a arquitetura será avaliada em um cenário cujo protocolo local industrial é o comumente usado Modbus TCP/IP, e o experimento será feito através da troca de dados

entre dispositivos industriais e um sistema de controle e monitoramento remoto via NG. A validação desta demonstração será feita através da criação de um ambiente industrial que será simulado para realizar a transmissão de dados de seus ativos ao computador de monitoramento remoto, atendendo a todos os requisitos do modelo RAMI 4.0. Além disso, serão realizadas análises do consumo de memória e energia para otimizar o projeto lidando com o mesmo através de seu modo mais econômico, trabalhando com a menor quantidade de recursos possíveis para a solução. Para isso, serão implementados testes relacionados à taxa de entrega e perda de pacotes durante a troca de informações entre ativos e sistema remoto, sendo avaliados em diferentes modos de economia de energia dos microcontroladores. Por fim, será realizada uma análise qualitativa dos benefícios da NG como proposta ao RAMI 4.0, para comparar o projeto proposto com as camadas de arquitetura do modelo de referência afim de validar sua viabilidade para a aplicação.

1.2 Organização do Texto

Este trabalho está dividido em 5 Capítulos. O Capítulo 1 possui o objetivo de introduzir o leitor ao assunto, de forma a construir uma base para as informações que serão discutidas durante o estudo. Neste Capítulo é evidenciado o cenário atual da Quarta Revolução Industrial e uma breve introdução das tecnologias emergentes que serão o foco principal deste estudo (ICPS, IIoT E FI), mostrando o objetivo do trabalho proposto e sua metodologia de avaliação.

No Capítulo 2 é fornecido ao leitor todo o embasamento teórico necessário para a contextualização do trabalho realizado. Neste capítulo, são iniciados os conceitos de Indústria 4.0 e suas tecnologias emergentes. Dentre essas tecnologias, IIoT e ICPS são abordados de maneira mais detalhada, descrevendo suas características e o modelo de referência de arquitetura RAMI 4.0. Também são destacados protocolos industriais e de *Internet of Things* (IoT), em português, Internet das Coisas, como *Message Queue Telemetry Transport* (MQTT), em português, Transporte de Telemetria de Fila de Mensagens, *Constrained Application Protocol* (CoaP), em português, Protocolo de Aplicativo Restrito, e o protocolo usado neste trabalho, denominado Modbus TCP/IP. Vista a dificuldade de se alterar toda a infraestrutura fabril para se adaptar aos requisitos de Indústria 4.0 e tecnologias emergentes, tem-se como solução apresentada o conceito de FI e da arquitetura NG, que será usada para a implementação do ICPS.

O Capítulo 3 consiste em uma revisão bibliográfica de trabalhos relacionados a ICPSs e a IIoT, comparando os projetos escolhidos e analisando suas respectivas vantagens, desvantagens, limitações e contribuições para a comunidade industrial.

No Capítulo 4 é apresentado o cenário de testes utilizado para a validação da dissertação. Primeiramente, são apresentados os equipamentos de *hardware* e *software*, seguidos pelo diagrama completo do cenário e descrição do funcionamento dos códigos de programação utilizados para a aplicação.

O Capítulo 5 fornece os resultados obtidos no cenário de testes da aplicação, além de uma

análise baseada na metodologia de validação, avaliando a eficácia do projeto. Os experimentos mostram que a NG é uma alternativa ao *status quo* em I4.0, trazendo novidades importantes para essa área, podendo desenvolver um ICPS baseado no modelo RAMI 4.0 com o diferencial de uma arquitetura FI capaz de lidar com as limitações da Internet atual.

Visto que o foco deste trabalho é avaliar a viabilidade da NG no desenvolvimento de um ICPS baseado no RAMI 4.0, o Capítulo 6 propõe uma análise qualitativa dos benefícios da NG como proposta ao modelo RAMI 4.0, comparando o sistema proposto com as camadas de arquitetura do modelo de referência.

No Capítulo 7, são apresentadas as considerações finais desta dissertação, tomando como base os experimentos realizados, concluindo sob quais circunstâncias a aplicação tem maior eficiência. Ainda, apresenta-se desafios futuros e possíveis propostas de continuidade para este estudo.

Por fim, são apresentadas as referências bibliográficas utilizadas.

Capítulo 2

Fundamentação Teórica

Esse capítulo cobre os aspectos necessários para o bom entendimento do trabalho apresentado nessa dissertação. A fim de garantir uma boa compreensão do que está sendo proposto, primeiramente serão apresentados os conceitos e pilares da Indústria 4.0 e suas principais tecnologias, com ênfase em IIoT e ICPSs, que são o foco deste trabalho. Também será introduzida a arquitetura de referência RAMI 4.0 como modelo escolhido para esta aplicação e o protocolo Modbus TCP/IP que é comumente usado em ambiente industrial Finalmente, devido às limitações da Internet atual e à dificuldade de mudar toda a infraestrutura fabril para suportar I4.0 e tecnologias emergentes, será apresentada como solução o uso da arquitetura NG no contexto de I4.0. Tal solução é avaliada como alternativa a arquitetura RAMI 4.0 com TCP/IP.

2.1 Indústria 4.0

No século XXI, a Indústria 4.0 está sendo desenvolvida para revolucionar o processo de manufatura ao reunir um conjunto de pilares que possibilitam a fusão dos mundos físico, digital, humano e biológico, além de novas tecnologias no ambiente industrial. A Figura 2.1 ilustra os principais pilares da Indústria 4.0.

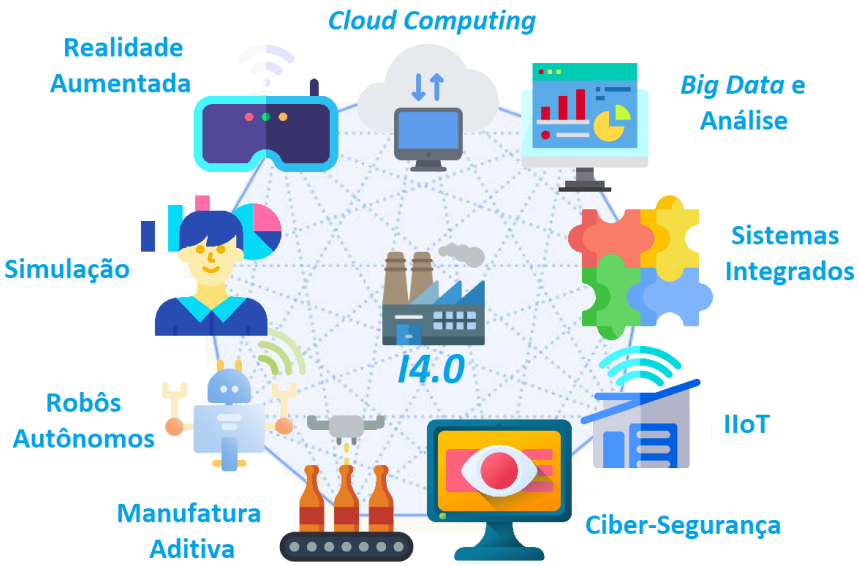

Figura 2.1: Principais Pilares da Indústria 4.0.

Este conjunto de pilares é formado por *Cloud Computing* (CC) [21], [22], Big Data e Análise de Dados [23], Sistemas Integrados [24], IIoT [25], [26], Ciber-Segurança [27], Manufatura Aditiva [28], Robôs Autônomos [29], Simulação [30] e Realidade Aumentada [31]. A combinação desses pilares é capaz de criar tecnologias essenciais para a Indústria 4.0, como por exemplo o CPS, que é o tema-chave desta pesquisa [32], [33].

A Indústria 4.0 pode ser entendida como uma rede inteligente de máquinas e processos industriais com a ajuda de Tecnologias de Informação e Comunicação (TIC) usadas em fábricas para produção flexível, logística otimizada, soluções orientadas para o cliente, melhor uso de dados e economia circular eficiente de recursos [34].

Desde a década de 1970, a tecnologia da informação tem sido incorporada ao ambiente industrial. A digitalização da produção consequentemente tem aumentado o nível de *Quality Of Service* (QoS), em português, Qualidade de Serviço, e facilitado a integração global de tecnologias como *Internet of Things* (IoT) [35], *Machine-to-Machine* (M2M) *communication*, em português, comunicação máquina-a-máquina, e processos de serviço de manufatura, que estão se tornando cada vez mais inteligentes [34].

Apesar de todos os benefícios e vantagens da implementação da Indústria 4.0, a proposta é bastante complexa, visto que a grande quantidade de digitalização e *networking* das empresas envolvidas aumenta o número de arquiteturas I4.0 criadas por diferentes autores e, consequentemente, possíveis problemas relacionados às redes de comunicação e interoperabilidade de sistemas [34]. Além disso, a digitalização das fábricas implica em um aumento significativo de ativos industriais digitais, e consequentemente em discussões envolvendo as limitações da Internet atual e ciber-segurança.

Baseado nas informações descritas acima, os Subcapítulos seguintes tem o intuito de

apresentar de forma mais detalhada as tecnologias IIoT e ICPS presentes na Indústria 4.0, visto que elas serão o foco desta pesquisa, e detalhar os principais modelos de referência de arquitetura ICPSs criados pela comunidade industrial, afim de garantir a interoperabilidade entre diferentes sistemas.

2.1.1 Industrial Internet of Things

IIoT, em português, Internet das Coisas Industrial, é um conceito também conhecido como *Internet Industrial*, termo criado pela *General Electric* (GE), *Internet of Everything*, nomeado pela Cisco, Internet 4.0, entre outros [36]. Ela é considerada uma subcategoria de IoT, em português, Internet das Coisas, para possibilitar a produção digital e as chamadas *Smart Factories* por meio da convergência de redes industriais e do conceito de informação digital.

A Internet Industrial é uma rede de dispositivos industriais composta por sensores, robôs e atuadores industriais complexos conectados a tecnologias de comunicação que possibilitam aos sistemas monitorar, analisar, entregar, coletar e alterar dados de forma rápida e fácil [37]. Esse conceito se disseminou no cenário industrial à medida que a digitalização das indústrias tem se tornado uma prioridade para as empresas. Como benefício, a IIoT oferece maior conectividade por meio da integração de máquinas, sensores, *middleware*, *software*, computação em nuvem e sistemas de armazenamento.

Semelhante à proposta IIoT, atualmente o conceito M2M existente é comumente implementado nas indústrias [38]. No entanto, a escala de operações IIoT é muito maior do que as encontradas nos ambientes industriais atuais, visto que a Internet Industrial permite uma análise online rápida e avançada de fluxos de dados usando hospedagem em nuvem por meio de *Big Data* [39].

IIoT, combinada com comunicação M2M e análise industrial de *Big Data*, pode gerar uma série de benefícios aos ambientes industriais, como por exemplo redução de operações; melhor desempenho e uso de ativos; redução do custo de ativos; tomada de decisão mais rápida; compra e venda de produtos como serviços, ampliando as oportunidades de negócios [40].

Dentre as principais tecnologias e conceitos de IIoT, pode-se citar *Digital Twins* [41], em português, Gêmeos Digitais, *Intelligent Edge* [42], em português, Borda Inteligente, Manutenção Preventiva [43] e *Radio-Frequency Identification* (RFID), em português, Identificação de Radiofrequência [44], [45]. *Digital Twins* é a criação de um modelo computadorizado de um objeto, possibilitando por meio de seu estudo em domínio virtual o entendimento e previsão de seu comportamento no mundo real, bem como a implantação de soluções para problemas antes que eles ocorram. *Intelligent Edge* é um sistema no qual os dados podem ser gerados, analisados e interpretados de forma rápida e com mais segurança. A Manutenção Preventiva é a análise e armazenamento dos dados de um ativo em um banco de dados para comparação de eventos, eliminando manutenções desnecessárias. RFID é um sistema que envolve etiquetas e leitores. No caso, os leitores identificam as etiquetas RFID por meio de

ondas de rádio, permitindo a leitura simultânea em distâncias razoáveis. As etiquetas RFID permitem o rastreamento e monitoramento de ativos de forma mais fácil e rápida. Essas tecnologias descritas acima contribuem para a criação de eficiências operacionais revolucionárias e a inovação de modelos de negócios no cenário IIoT.

Os principais desafios para a implementação de IIoT na Indústria 4.0 são a interoperabilidade entre os dispositivos, suas entidades virtuais, redes de comunicação, serviços e camada de inteligência. Complementar à Internet Industrial, o conceito de ICPS também é bastante discutido no ambiente fabril e sua abordagem será apresentada no Subcapítulo a seguir.

2.1.2 Sistema Ciber-Físico Industrial

Considerada uma das principais tecnologias presentes no cenário I4.0, o CPS, em português, Sistema Ciber-Físico, foi proposto pela cientista americana *Hellen Gil* em 2006 na *National Science Foundation* [46]. Em ambientes industriais, este termo é conhecido como ICPS, em português, Sistema Ciber-Físico Industrial, e aborda conceitos também presentes em IIoT. Como pode ser observado, há muitas superposições entre ambos os temas e inúmeras definições, mas são similares. Pode-se dizer que existe uma forte relação entre IIoT e ICPS. De fato, o ICPS é uma manifestação física da IIoT. Portanto, benefícios e desafios associados à IIoT também podem ser aplicados no cenário ICPS [47].

Enquanto sistemas IIoT focam mais na unificação e distribuição de dispositivos em uma rede industrial única via Internet, os ICPSs se concentram no ativo em específico, através de sua digitalização e tratamento de dados virtual que implicam em ações no mundo físico. Por isso, os ICPSs são responsáveis pela ligação entre os espaços virtuais e a realidade física, através da integração de networking, informática e armazenamento, possibilitando um ambiente industrial interativo com o desenvolvimento de *Smart Factories* [48].

Ao contrário dos sistemas embarcados tradicionais, seu foco principal é a criação de uma rede de dispositivos em massa para I4.0 [49]. Portanto, o ICPS consiste em uma unidade de controle capaz de lidar com sensores e atuadores que interagem com o mundo físico, processando os dados obtidos e trocando-os com outros sistemas e/ou serviços em nuvem através de uma interface de comunicação. Em outras palavras, os ICPSs podem ser vistos como sistemas capazes de enviar e receber dados de dispositivos por meio de uma rede.

Uma característica importante de um ICPS é sua capacidade de obter informações e serviços em tempo-real, independentemente de sua localização [50], implementando o acesso à Internet nas máquinas de manufatura [51]. Além da comunicação em tempo-real, é necessário garantir sua estabilidade, confiabilidade, eficiência e segurança nas operações [52]. Para isso, um dos objetivos em I4.0 é fornecer um suporte de segurança de alto nível em todas as camadas da arquitetura ICPS, protegendo informações confidenciais, ao mesmo tempo em que fornece anonimato de dados [53].

Apesar do foco deste trabalho ser voltado para Sistemas Ciber-Físicos no ramo industrial, ou simplesmente ICPS, é importante ressaltar que os CPSs atualmente são aplicados a outras áreas. Além do setor de manufatura [54], pode-se citar a área da saúde [55], energia

[56], *Smart Buildings* [57], transporte [58], áreas agrícolas [59], e rede de computadores [60], [61]. Na área de manufatura, é utilizado para o auto-monitoramento, controle da produção e compartilhamento de informações em tempo-real. Na área da saúde, pode ser utilizado no monitoramento remoto em tempo-real das condições físicas dos pacientes. No setor de energias renováveis, os sensores permitem o monitoramento e controle da rede, garantindo confiabilidade e eficiência no consumo de energia. Na área de *Smart Building*, a interação entre CPSs e dispositivos inteligentes pode reduzir o consumo de energia e aumentar a proteção, segurança e conforto para os moradores. Na área de transportes, esta tecnologia permite a comunicação entre os veículos e a infraestrutura, partilhando informações como intensidade de tráfego, localização de congestionamentos e acidentes para preveni-los. No setor de Agricultura, informações sobre o clima e recursos como dados de irrigação e umidade podem ser coletados, aumentando a precisão nos sistemas de gerenciamento agrícola. Por fim, nas redes de computadores, este conceito é aplicado para o melhor entendimento dos sistemas e comportamentos dos usuários em ambientes virtuais.

Na Indústria 4.0, os ICPSs cobrem não apenas máquinas e produtos, mas também clientes, prestadores de serviços e estoques, garantindo uma interação adequada em todas as áreas sendo executadas de forma autônoma [62].

Com a quarta revolução industrial e o aumento dos dados e dispositivos industriais, propostas e modelos de arquiteturas para ICPSs surgiram, como: a Arquitetura 5C criada em 2014 por Lee et al. e amplamente usada no desenvolvimento de CPSs para sistemas embarcados e infraestruturas industriais de pequeno e médio porte [63]; o *Industrial Internet Reference Architecture* (IIRA), desenvolvido em 2015 pela *Industrial Internet Consortium* (IIC) para a integração e cooperação entre indústrias com foco em IIoT [64]; e o RAMI 4.0, focado no setor de manufatura para virtualização de dispositivos na cadeia de valor. Devido ao seu propósito chave citado anteriormente, a arquitetura de referência RAMI 4.0 será escolhida neste trabalho como modelo para o projeto ICPS, sendo descrita no Subcapítulo a seguir.

2.1.3 RAMI 4.0

Desenvolvido em 2015 pela *Platform Industrie* 4.0, RAMI 4.0 é um Modelo de Referência de Arquitetura I4.0 criado para definir estruturas de comunicação e uma linguagem comum dentro da fábrica com seu próprio vocabulário, semântica e sintaxe. Tal linguagem permite a integração de IoT e serviços no contexto I4.0, conectando-os ao resto do mundo [65]. Ela consiste em uma SOA que combina componentes de Tecnologias de Informação (TI) para promover os principais aspectos de I4.0, como a integração horizontal entre redes de fábricas e plantas; e a integração vertical dentro de uma fábrica, com produtos conectados entre si.

Esta arquitetura é representada por um mapa tridimensional composto por 3 eixos, denominados Níveis de Hierarquia, Ciclo de Vida do Produto e Camadas de Arquitetura, que abordam questões da Indústria 4.0 de forma estruturada, garantindo que todos os participantes envolvidos na fábrica possam se entender e conectando todo o processo de fabricação.

A Figura 2.2 mostra o modelo tridimensional do RAMI 4.0 [66].

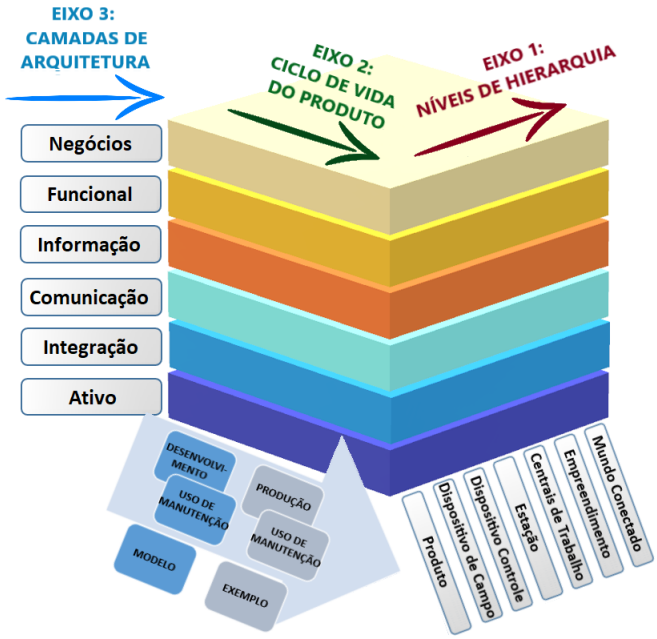

Figura 2.2: Modelo Tridimensional do RAMI 4.0.

O Eixo 1, responsável pelos Níveis de Hierarquia, pretende mudar a ideia proposta da Indústria 3.0, na qual a infraestrutura era baseada em hardware especializado, limitando suas funções; o modelo de comunicação baseado na hierarquia; e os produtos eram visto como isolados um do outro. No cenário I4.0, o objetivo é disseminar a ideia de máquina e sistemas flexíveis; funções distribuídas pela rede, garantindo a interação e comunicação entre todos os participantes envolvidos; e produtos sendo vistos como parte da arquitetura.

O Eixo 2, responsável pelo Ciclo de Vida do Produto, descreve os ativos da cadeia de valor desde a sua ideia inicial e desenvolvimento até a sua produção, utilização e manutenção final. É válido ressaltar que ativos são objetos que têm valor para uma organização, como um dispositivo ou equipamento [67].

O Eixo 3 é responsável pelo desenvolvimento da proposta ICPS, que é o foco deste trabalho, e será descrito de forma detalhada a seguir. Ela engloba as Camadas de Arquitetura do RAMI 4.0, sendo definidas como [67]:

1. **Camada Ativo:** A Camada Ativo representa as coisas físicas do mundo real. Essas "coisas"podem ser componentes, *hardware*, documentos e trabalhadores.

2. **Camada de Integração:** A Camada de Integração é responsável pela transição do mundo real para o virtual. Representa os ativos físicos e suas capacidades digitais, consequentemente proporcionando o controle via computador, possibilitando a geração de eventos para si.

3. **Camada de Comunicação:** A Camada de Comunicação fornece comunicação padronizada de serviços e eventos ou dados para a Camada de Informação, e de serviços e comandos de controle para a Camada de Integração. Essa camada do RAMI 4.0 está focada no mecanismo de transmissão, sendo responsável pela descoberta das redes e pela conexão entre elas. A fim de garantir a interoperabilidade e o uso de padrões para sua implementação, a Camada de Comunicação foi mapeada para o Modelo em Camadas da *International Organization for Standardization/Open System Interconnection* (ISO/OSI) e foi organizada com diferentes padrões e tecnologias. Na Camada Física, as tecnologias com fio e sem fio são especificadas; a Camada de Enlace de Dados inclui Ethernet, Wi-Fi e *Global System for Mobile* (GSM)/4G; a Camada de Rede engloba IP e a *Internet Protocol Security* (IPSec); a Camada de Transporte consiste nos protocolos *Transmission Control Protocol* (TCP), em português, Protocolo de Controle de Transmissão, e *User Datagram Protocol* (UDP), em português, Protocolo de Datagrama do Usuário; finalmente, nas camadas de 5 a 7 (Sessão, Apresentação e Aplicação), tem-se a utilização do OPC UA. Em geral, a comunicação de nuvem-para-nuvem e empresa-para-nuvem é bastante discutida na comunidade I4.0, bem como as tecnologias emergentes *Time-Sensitive Networking* (TSN) [68] e 5G [69].

4. **Camada de Informação:** A Camada de Informação descreve os serviços e dados que podem ser oferecidos, utilizados, gerados ou modificados pela funcionalidade técnica do ativo.

5. **Camada Funcional:** A Camada Funcional é responsável pela descrição das funções lógicas de um ativo, como sua funcionalidade técnica, no contexto de I4.0.

6. **Camada de Negócios:** A Camada de Negócios organiza os serviços para criar processos de negócios e conectá-los entre si, apoiando modelos de negócios sob restrições legais e regulatórias.

O eixo vertical das Camadas de Arquitetura tem como objetivo descrever as entidades físicas da rede industrial em modelagem, como dispositivos, equipamentos e máquinas, e mapeá-las para suas respectivas representações virtuais como um *Industry* 4.0 *Component* (I4.0C), em português, Componente da Indústria 4.0, capaz de descrever detalhadamente as propriedades de um ativo.

I4.0C são objetos globalmente e unicamente identificáveis com capacidade de comunicação [65], sendo associado e representado por um ativo e um *Administration Shell* (AS), que contém informações relevantes para a gestão do ativo. Inclui também as funcionalidades técnicas dos ativos, armazenando todos os dados e informações sobre os mesmos. AS é a interface padronizada para redes de comunicação, capaz de conectar as coisas físicas à Indústria 4.0. O I4.0C pode ser relacionado a um equipamento, máquina ou produto na Camada de Ativos e o AS às Camadas de Informação, Funcional e de Negócios. Por exemplo, um ativo, como uma máquina, representa a parte física e o AS representa sua parte digital. Além disso, todos os ASs no sistema, que também podem ser tratados como gêmeos digitais, são gerenciados

por um *Superior System Administration Shell* (SAS), capaz de combinar a intercomunicação entre eles [70].

Baseado nas informações das arquiteturas descritas acima, o modelo RAMI 4.0 foi escolhido como base para o desenvolvimento deste trabalho, devido ao seu foco no setor de manufatura para criar um sistema de monitoramento e controle remoto através da digitalização de ativos industriais. Para a implementação prática deste modelo, a arquitetura FI NG que será detalhada a seguir é usada, visto que sua característica diferencial de uma SOA centrada à nomeação pode definir estruturas de comunicação, identificação única de ativos e uma linguagem comum dentro da indústria com o desenvolvimento de seu próprio vocabulário e semântica. Além desse suporte às premissas da arquitetura de referência, a definição NG centrada a nomes também realiza uma integração com as intenções baseada em linguagem, permitindo que as pessoas se expressem usando linguagem natural, se aproximando dos ideais discutidos na Indústria 5.0, e direciona suas ideias buscando soluções que exploram as limitações da Internet atual, sendo um problema a ser considerado no cenário industrial atual com a emersão de IIoT e ICPSs.

Ao se utilizar NG, o conceito de I4.0C definido pelo RAMI 4.0 pode ser desenvolvido, onde ativos podem ser unicamente e globalmente identificáveis e apresentar sua capacidade de descrição técnica, funcional, oferta de serviços e negócios de forma digital, através da utilização de *Name Bindings* (NB), em português, Ligação de Nomes. Além disso, a NG capacita a troca de mensagens de forma segura entre diferentes ativos únicos com a arquitetura embarcada, possibilitando, por exemplo, o monitoramento e controle remoto dos mesmos. Tendo em vista a complexa tarefa de mudança geral em setores de manufatura legados, a NG também pode ser implementada para permitir as ideias de I4.0 e tecnologias emergentes. Como exemplo, este projeto propõe sua arquitetura baseado no modelo RAMI 4.0 em ambientes fabris cujo protocolo de comunicação industrial local usado é o Modbus TCP/IP, que será descrito no próximo Subcapítulo. Em seguida, serão apresentados os conceitos de FI e da arquitetura NG.

2.2 Protocolos Industriais e de Internet of Things

A seguir, será realizada uma breve apresentação dos protocolos comumente usados em aplicações de IoT, como MQTT e CoAP. Além disso, já que para este trabalho foi adotado o uso do protocolo industrial Modbus TCP/IP‘, o mesmo será descrito posteriormente de forma mais detalhada.

2.2.1 MQTT e CoAP

O MQTT é um protocolo de mensagens desenvolvido idealmente para comunicação Machine-to-Machine (M2M), em português, Máquina-à-Máquina, e IoT, utilizando como padrão o Organization for the Advancement of Structured Information Standards (OASIS), em português, Organização para o Avanço de Padrões de Informação Estruturada. O transporte

de mensagens foi projetado através do conceito de publicação/assinatura. Devido à sua estrutura extremamente leve, este protocolo é ideal para conectar dispositivos remotos com um pequeno espaço de código e largura de banda de rede mínima. O MQTT hoje é usado em uma varieade de setores industriais, como por exemplo automotivo, manufatura, telecomunicações, petróleo e gás, etc. Visto que seus princípios arquitetônicos são otimizados para redes TCP/IP, foi criada a variação Message Queuing Telemetry Transport for Sensor Network (MQTT-SN), em português, Transporte de Telemetria de Fila de Mensagens para Redes de Sensores, que é uma variação do protocolo destinada a redes que não sejam baseadas neste princípo, como por exemplo a tecnologia sem fio ZigBee, que é um padrão global aberto para atender às necessidades exclusivas de redes IoT sem fio de baixo custo e baixo consumo de energia [71]. A Figura 2.3 ilustra o processo de publicação e assinatura de mensagens MQTT, onde é usado um *broker*, que é um servidor intermediário capaz de receber todas as mensagens publicadas e, em seguida, roteá-las para os clientes de destino relevantes.

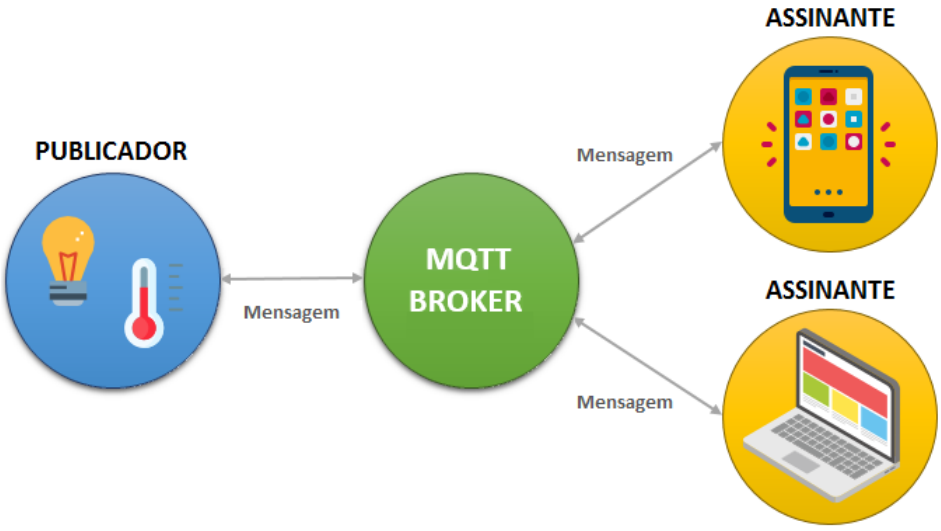

Figura 2.3: Processo de publicação e assinatura do protocolo MQTT.

O CoAP, por sua vez, é um protocolo de transferência *web* especializado para uso com nós restritos em redes da IoT, também projetado para comunicação M2M, como, por exemplo, aplicações em automação predial e energia inteligente. Este protocolo é semelhante ao HyperText Transfer Protocol (HTTP), em português, Protocolo de Transferência de Hipertexto, porém as iterações entre máquinas atuam nas funções de cliente e servidor. Assim sendo, um pedido CoAP similar ao do HTTP e é enviado por um cliente para solicitar uma ação em um servidor, que por sua vez envia uma resposta com um código. Diferente do HTTP, a transferência de dados do protocolo CoAP é realizada de forma assíncrona ao longo de um transporte orientado a datagramas, como o UDP, usando uma camada de mensagens que suporta uma confiabilidade opcional). Desta forma, o modelo de mensagens CoAP baseia-se na troca de mensagens UDP entre os pontos finais, se posicionando na arquitetura TCP/IP entre as camadas Aplicação e Transporte, criando duas subcamadas, denominadas

"Requisição/Resposta"e "Mensagem"[72]. A Figura 2.4 ilustra o posicionamento do protocolo CoAP na arquitetura TCP/IP.

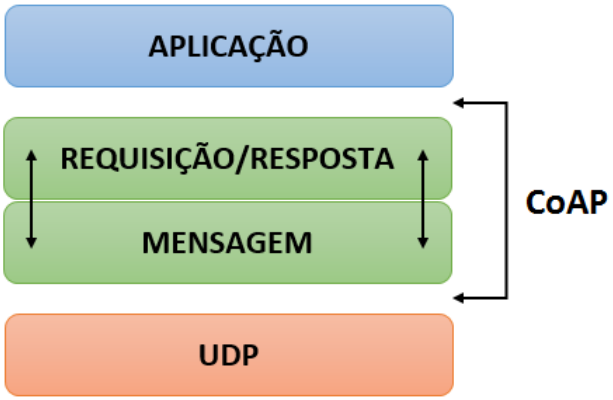

Figura 2.4: Posicionamento do protocolo CoAP na arquitetura TCP/IP.

2.2.2 Protocolo Modbus TCP/IP

Desenvolvido em 1979 pela Modicon, o protocolo Modbus foi incorporado em sistemas de automação industrial como um método padrão para transferência de informações de entradas e saídas (I/O) discretas e analógicas e registro de dados entre dispositivos de controle e monitoramento. A comunicação entre dispositivos Modbus é realizada pela técnica de mestre-escravo (cliente-servidor), no qual um cliente inicia as transações, chamadas de consultas, e os servidores respondem com os dados requisitados ao cliente ou tomando as ações requisitadas na consulta. Assim sendo, a definição dos dispositivos a serem configurados como Cliente e Servidor depende da aplicação proposta. Desta forma, clientes podem endereçar servidores individuais ou iniciar uma mensagem broadcast para todos, enquanto os servidores apenas respondem à todas as consultas endereçadas individualmente, mas não respondem as consultas *broadcast* [73].

Como pode-se perceber, o protocolo Modbus trata-se de um protocolo de aplicação, pois define regras para organizar e interpretar dados, tratando-se simplesmente de uma estrutura de mensagem independente da camada física. Dentre os formatos Modbus existentes, tem-se o Modbus ASCII, RTU e TCP/IP (ou simplesmente Modbus TCP). Por padrão, esses formatos são representados por um *Protocol Data Unit* (PDU), em português, Unidade de Dados de Protocolo e a camada de rede, que define a *Application Data Unit* (ADU), em português, Unidade de Dados de Aplicação. Como o foco deste trabalho será o formato Modbus TCP/IP, o mesmo será detalhado a seguir. Para isso, a Figura 2.5 ilustra a construção do pacote de dados Modbus TCP/IP [74].

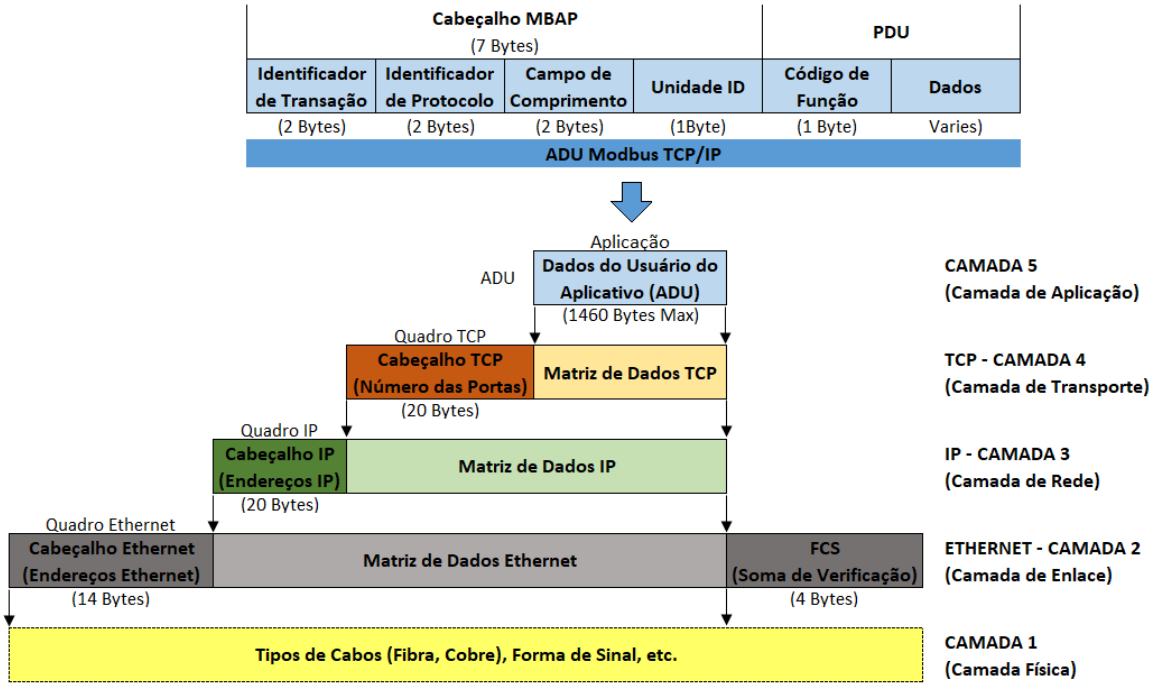

Figura 2.5: Construção de pacote de dados Modbus TCP/IP.

Como pode ser observado pela Figura 2.5, o quadro Modbus com a transmissão TCP é formada pela: PDU, que consiste em 1 *byte* para o Código da Função e um espaço reservado para os dados (o comprimento total da PDU é limitado a 253 *bytes*); e ADU, que engloba a PDU, e o cabeçalho do *Modbus Application Protocol* (MBAP), em português, Protocolo de Aplicação Modbus. O MBAP, por sua vez, é formado por 7 *bytes*, dos quais: 2 *bytes* são destinados ao Identificador de Transação, usada para pareamento de transações quando as últimas mensagens são enviadas pela mesma conexão TCP por um cliente sem esperar pela resposta anterior; 2 *bytes* referem-se ao Identificador de Protocolo, sendo definido como 0 para serviços Modbus; 2 *bytes* são alocados para o Comprimento do Campo para delinear o comprimento do pacote; e 1 *byte* para a Unidade ID, com o objetivo de identificar um servidor remoto localizado em uma rede diferente da TCP/IP para ponte serial (normalmente é configurado para 00 ou FF e ignorado pelo servidor) [73].

Desta forma, a aplicação do *host* (cliente) realiza sua solicitação, passando os dados para as camadas inferiores, que por sua vez adicionam suas informações de controle no pacote na forma de cabeçalhos/rodapés do protocolo até chegar na camada física, onde ele é transmitido eletronicamente ao destino (servidor). Então, o pacote sobe pelas diferentes camadas de seu destino, com cada uma decodificando sua respectiva porção da mensagem e removendo o cabeçalho/rodapé que foi inserido pela mesma camada do cliente, até finalmente alcançar a aplicação [73]. Para o trabalho proposto.

Em relação à PDU, composta pela Função do Código e Dados, existem definições devem ser conhecidas pelo usuário. Os modelos de dados Modbus podem ser definidos através do Bloco de Memória, Tipo de Dado, Acesso ao Cliente e Acesso ao Servidor, e seu en-

dereçamento é constituído por um Prefixo (P) de modo a identificar em qual bloco de memória o dado será alocado. O Acesso ao Cliente e Servidor permite ao usuário conhecer as ações que o mesmo pode exercer, como Leitura/Escrita (L/E) ou Somente Leitura (SL). A tabela 2.1 ilustra os modelos de dados Modbus e a forma de endereçamento de dados [74].

Tabela 2.1: Modelos de Dados Modbus.

Bloco de Memória	**Tipo de Dado**	**Acesso ao Cliente**	**Acesso ao Servidor**	**Exemplo de Uso**	**P**
Coils	Booleano	L/E	L/E	Uma saída digital	0
Entradas discretas	Booleano	SL	L/E	Uma entrada digital	1
Registradores de Entrada	Palavra não sinalizada	L/E	L/E	Várias saídas digitais ou um parâmetro de configuração	3
Registradores Holding	Palavra não sinalizada	SL	L/E	Várias entradas digitais ou uma entrada analógica	4

A especificação define que cada bloco contém um espaço de endereçamento de até 65.536 elementos considerando os endereços estendidos, e cada bloco de memória aloca uma faixa desse espaço, porém todos os registradores não necessariamente precisam estar em uso. Com a definição da PDU, o Modbus define o endereço de cada elemento de dados na faixa de 0 a 65535. Sendo assim, baseado na Tabela 2.1 e no conceito de prefixo de endereçamento, o Registrador *Holding* 1, por exemplo, assume o endereço 4001, pois possui o prefixo 4. Porém, analisando desta maneira, o *Coil* 4001 também poderia ser definido com o endereço 004001, visto que seu prefixo é 0. Então, para evitar essas ambiguidades, adota-se por padrão a nomenclatura com o número do prefixo (P) seguido pela quantidade máxima de elementos com 5 dígitos, mesmo não sendo totalmente usada (PXXXXX). Desta forma, o Registrador *Holding* 1 assume o endereço 400001, enquanto o *Coil* 4001 assume o endereço 004001 [75]. A Figura 2.6 ilustra a nomenclatura dos endereços estendidos de cada Bloco de Memória.

Blocos de Memória: Nomenclatura dos Endereços Modbus

Blocos de Memória:	Coils			Entradas Discretas			Registradores Holding			Registradores de Entrada		
Nomenclatura dos Endereços (0 a 65535):	000001	...	065535	100001	...	165535	300001	...	365535	400001	...	465535

Figura 2.6: Nomenclatura dos endereços estendidos de cada Bloco de Memória.

Para a solução desenvolvida nesta dissertação, foi adotado o protocolo industrial Modbus TCP/IP para ser usado como referência no ambiente industrial.

2.3 Internet do Futuro

Conhecida como a rede global do século 21, a Internet provou ser uma tecnologia revolucionária, transformando uma rede inicialmente projetada para conectar pesquisadores acadêmicos em uma rede de comunicações global. Porém, as demandas crescentes de informações pela rede com a proliferação de dispositivos móveis e a explosão de dados na web vindas através do conceito IoT, dispositivos e sensores conectados à rede com RFID, estão evidenciando cada vez mais as limitações da Internet atual [18].

Dentre os desafios da Internet atual, destaca-se o endereçamento IP, visto que o tráfego de dados da Internet cresce exponencialmente, e a quantidade de dispositivos em rede cada vez maiores torna o modelo IPv4 ultrapassado pela possibilidade de falta de endereços no futuro. Além disso, não há um consenso sobre um cronograma de migração para seu sucessor IPv6 [18]. Devido à emersão de IIoT no cenário industrial e consequentemente de dispositivos industriais em massa conectados em rede, a limitação física de endereços IP é um problema a ser considerado em I4.0.

Outra questão a ser discutida refere-se à estrutura de identificação e localização dos elementos da rede, visto que o endereço IP atua como um Identificador (ID) e Localizador (LOC) de forma unificada. Em um ambiente móvel, por exemplo, no caso de mudança de *host*, o LOC, que indica o ponto de conexão com a Internet, também pode mudar. Em I4.0, o modelo RAMI 4.0 especifica a necessidade de identificação única de ativos industriais para comunicação e digitalização fabril, e o uso de endereçamento IP como identificador único remete a um conflito nesta definição, vista a possibilidade de alteração do mesmo com uma possível mudança de localização. Uma possível solução seria a separação entre ID para identificação e LOC para roteamento, e a alocação do ID ao próprio *host*, e não à sua interface [76].

Visto que a Internet atual assume a pilha de protocolo IP comum a todos os nós da rede, em um ambiente mais heterogêneo com redes de características bastante diversas, um mecanismo de entrega comum único como o roteamento IP pode não suportar efetivamente essa heterogeneidade. Para isso, deve-se pensar no uso de diferentes mecanismos de entrega, variando de acordo com as características da rede [76]. Pensando no cenário de I4.0 onde o futuro está direcionado à uma rede capaz de utilizar diferentes tecnologias e protocolos, podendo ser facilmente programável e adaptável às aplicações, é necessária uma arquitetura que suporte essa flexibilidade e heterogeneidade, sendo FI uma alternativa interessante para este atender a este requisito.

Diante dessas e outras limitações da Internet atual, surgiu a ideia do desenvolvimento da FI, em português, Internet do Futuro, que significa qualquer rede semelhante à Internet que possa vir a emergir. O propósito de FI é adotar abordagens evolutivas, onde os protocolos fundamentais da Internet atual possam ser mantidos e novas ideias possam ser introduzidas de forma incremental. Além disso, também existem abordagens revolucionárias, nas quais pensa-se no redesenho da arquitetura de Internet do princípio [77], ou folha em branco, em inglês, *clean slate*.

Dentre as abordagens revolucionárias que surgiram para contornar as limitações da Internet atual, podem ser destacadas: espaços de nomeação e resolução de nomes ilimitados [78]; mobilidade do nó [79]; identificação e distribuição eficiente de conteúdo na rede [80]; *Self-Verifying Name* (SVN), em português, Nomeação Auto-Certificável, para fornecer redes intrinsecamente seguras [81]; empilhamento de protocolo dinâmico e auto-organizado [82]; e controle e operação de rede baseada em *software* [83], [77].

No contexto de Arquiteturas FI, os seguintes paradigmas são empregados: SOA [84]; *Information-Centric Networking* (ICN), em português, Rede Centrada em Informações [85]; *Software-Defined Networking* (SDN), em português, Rede Definida por *Software* [86]; *Service-Centric Networking* (SCN), em português, Rede Centrada em Serviços [87]; *Network Function Virtualization* (NFV), em português, Virtualização de Função de Rede [88]; *Self-verifying Names* (SNV), em português, Nomes Auto-Verificáveis [89]; Desacoplamento de ID/LOC [90]; e resolução de nome distribuído [91], [77].

Vista a emersão do uso da Internet Industrial para o conceito de Indústria 4.0 devido à IIoT e digitalização das fábricas, e consequentemente aos problemas descritos nas subseções anteriores relacionados à suas limitações, uma arquitetura de Internet do Futuro desenhada de acordo com o modelo RAMI 4.0 é ideal para o desenvolvimento dos ICPSs. Sendo assim, a NG é uma alternativa de arquitetura FI para suportar as características deste modelo de referência através de sua estrutura de nomeação que permite a digitalização de ativos, além de se preocupar e lidar com as limitações da Internet atual. Em adição à essas premissas, a NG tem como diferencial a capacidade permitida ao usuário de se expressar em linguagem natural, aproximando o homem à máquina, o que caracteriza um direcionamento da arquitetura aos conceitos primários de Indústria 5.0. Considerando IoT como o ingrediente-chave de seu design, a NG é capaz de integrar os paradigmas de rede revolucionários e evolucionários e criar um ICPS que atenda aos requisitos do modelo de referência RAMI 4.0 de forma inovadora através de sua arquitetura centrada a nomes para identificar unicamente os ativos industriais. A arquitetura NG será detalhada a seguir.

2.3.1 NovaGenesis

NG é uma proposta de arquitetura FI centrada a nomes, serviços e informações que integra redes e *clouds*, garantindo a troca, processamento e armazenamento de informações através da integração de uma variedade de conceitos emergentes. Desta forma, serviços se organizam com base em nomes, NBs e contratos para cumprir metas através de uma estrutura semanticamente rica. Como exemplo, essa característica permite a implementação de protocolos como serviços para convergir redes sem fio e cabeadas [20].

Tratando da pilha de protocolos baseada no modelo ISO/OSI, existem implementações da arquitetura sobre a camada de transporte com o uso dos protocolos TCP e UDP, tornando-se uma *web* da Internet corrente. Atualmente, foram realizadas melhorias na arquitetura, tendo sua última versão usando *Raw Socket* como nova proposta, onde a camada de transporte não é usada, realizando a comunicação diretamente através da camada de enlace e,

consequentemente, não havendo retransmssão no caso de perda de mensagens. Para o desenvolvimento desta dissertação de mestrado, será adotada o uso da NG em sua nova versão, com a comunicação direta via camada de enlace.

A NG foi desenvolvida baseada princípios com abordagens revolucionárias para lidar com os problemas relacionados às limitações da Internet atual mencionadas na Seção 2.3. Dentre esses princípios, podem-se destacar: Nomeação; Resolução Ilimitada de Nomes; Desacoplamento de ID/LOC; Mobilidade; Serviços, Protocolos e Pilha de Protocolos; Operação e Gerência; e Encaminhamento e Roteamento [20].

O princípio NG de Nomeação propõe um serviço de resolução de nomes genérico. Neste contexto, todas as entidades (domínios; computadores; *Operating System* (OS), em português, Sistema Operacional; processos; e nomes de conteúdos) podem ser nomeadas na forma de *Natural Language Names* (NLNs), em português, Nomes de Linguagem Natural; e/ou SVNs. Além disso, há a adoção de NBs cujo objetivo é ligar diferentes nomes formando um grafo de nomes distribuído e seguro, que é armazenado em tabelas *hash* distribuídas. No conceito de Indústria 4.0 e IIoT, onde existe a necessidade de implementação de dispositivos como objetos únicos e identificáveis, o princípio de nomeação é muito importante, e SVNs podem ajudar na privacidade de dados dos dispositivos pela sua característica de verificação de integridade implícita [92]. A Figura 2.7 ilustra um grafo de ligação de nomes. Assim sendo, cada ligação de nomes representa o relacionamento entre as entidades, tendo um operador semântico associado.

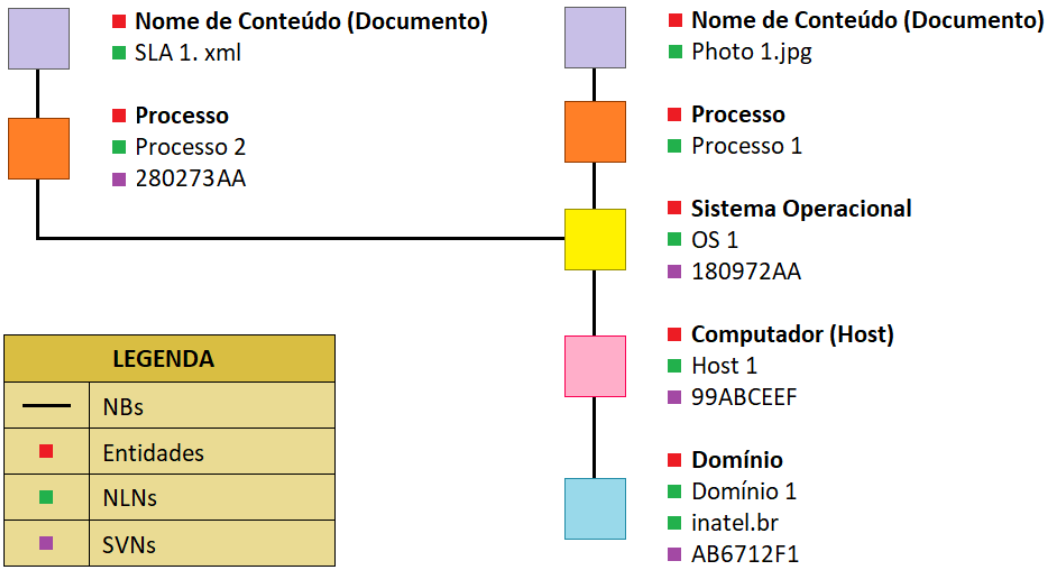

Figura 2.7: Exemplo de um Grafo de Ligação de Nomes.

Como mostrado na estrutura do grafo de nomes da Figura 2.7, os NBs permitem que as entidades possam ser representadas, como domínios e objetos de informação, recebam NLNs e/ou SVNs únicos, e sejam capazes de estabelecer relações com outras entidades. Para citar como exemplo, pode-se dizer que os operadores semânticos usados na Figura 2.7 são

"contém"e "está contido". Desta forma, pode-se dizer que Domínio 1 de NLN "inatel.br"e SVN "AB6712F1"está contido no *host*, cujo NLN é "Host 1"e seu respectivo SVN é "99ABCEEF", e, da mesma forma, que o "Host 1"contém o "Domínio 1". Como outros exemplos a serem considerados, também pode-se dizer que o documento de NLN "Photo 1.jpg"está contido no "Processo 1". Além disso, pode-se observar também que o Sistema Operacional de NLN "OS 1"contém o Processo de NLN "Processo 1". Desta forma, o princípio NG de Resolução Ilimitada de Nomes também é aplicado, uma vez que os NBs publicados permitem que qualquer nome relacionado possa ser resolvido em outros nomes [92].

Além dos princípios destacados anteriormente, pode ser observada na Figura 2.7 o Desacoplamento de ID/LOC. Visto que os nomes são usados para denotar escopos e espaços, os nomes únicos em um escopo são usados como ID, enquanto nomes usados para dar noção de distância em um espaço são usados como endereços (LOC). Esta característica consequentemente garante o princípio de Mobilidade via religação de nomes, já que o grafo propõe nomes que identificam uma entidade e nomes que a localizam, separadamente. Assim, uma entidade que se move mantém seu ID, mas muda sua ligação com nomes que a localizam [92].

O princípio de Serviços, Protocolos e Pilha de Protocolos define que todo *software* é um serviço, incluindo as implementações de protocolos, e toda a operação é baseada em contratos. Desta forma, seu ciclo de vida pode ser descrito como: exposição de características e capacidades; descoberta; negociação; contrato; monitoramento de qualidade; reputação; e liberação de recursos. No caso dos protocolos, são implementados em terminais como serviços, concorrendo entre si para ofertá-los. A pilha de protocolos, por sua vez, é formada dinamicamente através de contratos entre protocolos construída em tempo de execução [92].

O princípio de Operação e Gerência opera via serviços representantes das "coisas", permitindo que os recursos físicos, definidos por *software*, sejam representados por serviços que estabelecem contratos em seu nome. Dessa forma, as implementações de protocolos devem contratar recursos de hardware através de seus respectivos representantes [92].

Por fim, o princípio de Roteamento e Encaminhamento é baseado na descoberta de trajetos através da resolução recursiva de identificadores em localizadores. Desta forma, este princípio é similar ao Domain Name Service (DNS), em português, Sistema de Nomes de Domínio, porém cada domínio possui seu sistema de nomes e cache de rede, e a lógica de consulta é baseada em contratos. Uma tabela nos nós da rede (*hardware*) é preenchida com NBs fornecidos pelos representantes de *software* (serviço representante). Se um comutador ou roteador não sabe o que fazer com um quadro ou pacote, ele é direcionado ao seu serviço representante, que por sua vez configura a ligação no *hardware* [92].

A seguir, será realizada a descrição geral do projeto NG, que por sua vez está dividida em: estrutura de mensagens e serviços; serviços de publicação/assinatura; serviços de rede; serviços de domínio; e implementação.

NovaGenesis: Estrutura de Mensagens e Serviços

Como estrutura de mensagens, o design NG utiliza componentes arquitetônicos que são divididos em dois grupos: processadores de informação, que inclui *Ação*, *Bloco* e*Processo*; e objetos de informação, estruturado com *Linhas de Comando (LC)*, *Payload* e *Mensagem*. Desta forma, um objeto de informação é definido como uma *Mensagem* formada por um conjunto de *Linhas de Comando* e o *Payload*. Por outro lado, um processador de informação é um *Processo* que armazena todas as mensagens que estão sendo processadas, e pode ser definido como um serviço composto por uma série de *Blocos*. Cada *Bloco* é formado por um conjunto de *Ações*, que são responsáveis por processar e analisar as *Linhas de Comando* das *Mensagens*. Dentre os objetos de *Blocos*, são classificados os *Blocos* especializados, cuja funcionalidade depende do serviço; e os chamados *Blocos* comuns, que são aqueles instanciados em todos os processos. Como exemplo de *Blocos* comuns, pode-se citar o *Gateway* (GW) e a *Hash* Table (HT), em português, Tabela *Hash*. O GW é responsável pela comunicação entre *Blocos* dentro de *Processo* e pela comunicação entre *Processos* dentro de um OS, enquanto o HT é responsável pelo armazenamento de NBs usando uma estrutura de dados de tabela *hash* [82]. A Figura 2.8 ilustra uma estrutura de mensagens e serviços NG e a relação entre os processadores e objetos de informação, mostrando a visão interna geral de um *Processo*.

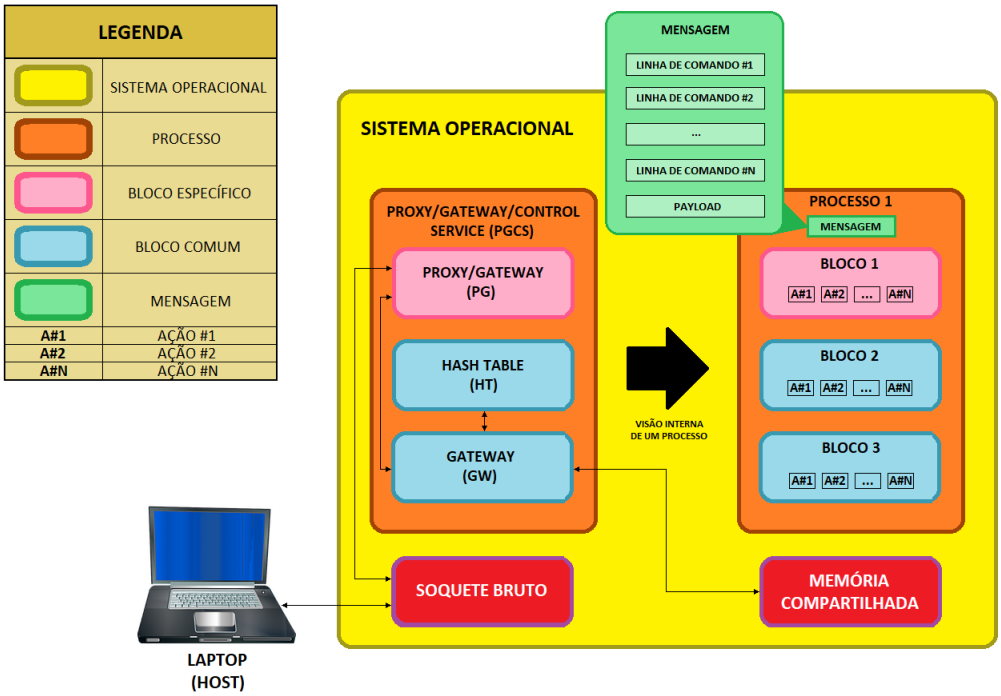

Figura 2.8: Estrutura de Mensagens e Serviços.

As Linhas de Comando NG são padronizadas de acordo com a Figura 2.9.

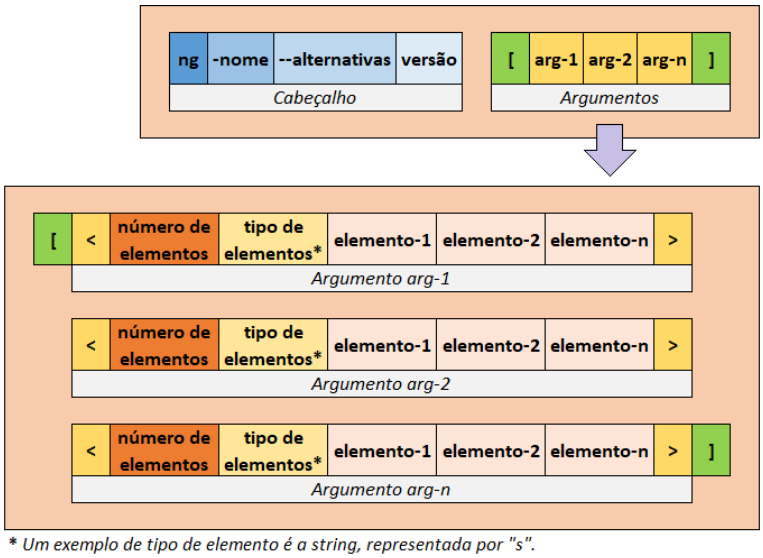

Figura 2.9: Estrutura da Linha de Comando NG.

Baseada na Figura, 2.9, podem-se destacar as principais Linhas de Comando necessárias para a construção da mensagem e funcionamento básico da NG. Assim sendo, as mensagens relacionadas às tarefas do Ciclo de Vida NG podem ser resumidas por: *Linha de Comando de Roteamento*, utilizada inicialmente em todas as mensagens NG; Linhas de Comando particulares à cada tarefa; *Linha de Comando de Verificação*; e *Linhas de Comando de Tipo e Sequência de Mensagem*, implementadas como controle nas mensagens NG (com exceção da mensagem *Hello*). A Figura 2.10 ilustra a estrutura de mensagem geral NG, além das Linhas de Comando essenciais mencionadas anteriormente para a NG embarcada.

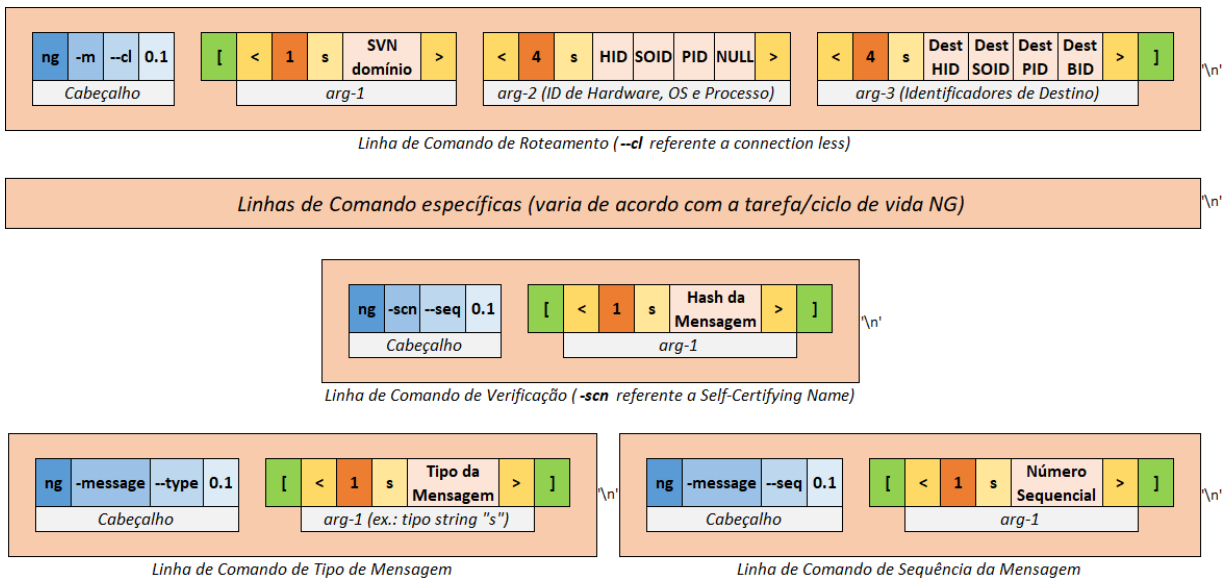

Figura 2.10: Construção da Mensagem NG e Linhas de Comando NG essenciais para NG embarcada.

A *Linha de Comando de Roteamento* é responsável por expôr seus SVNs aos seus pares e determinar o destino da mensagem. A *Linha de Comando de Verificação* assina a mensagem com seu SVN. A *Linha de Comando de Tipo de Mensagem* é auto-explicativa, e pode-se exemplificar o Tipo 1, referente à Mensagem do Usuário. Por fim, a *Linha de Comando de Sequência de Mensagem* indica o número sequencial da mensagem.

Já as Linhas de Comando particulares de cada tarefa, também conhecidas como Linhas Específicas, são responsáveis pelos estados do ciclo de vida NG e complementam a estrutura da Mensagem NG embarcada da Figura 2.10. A Figura 2.11 detalha as Linhas de Comando Específicas.

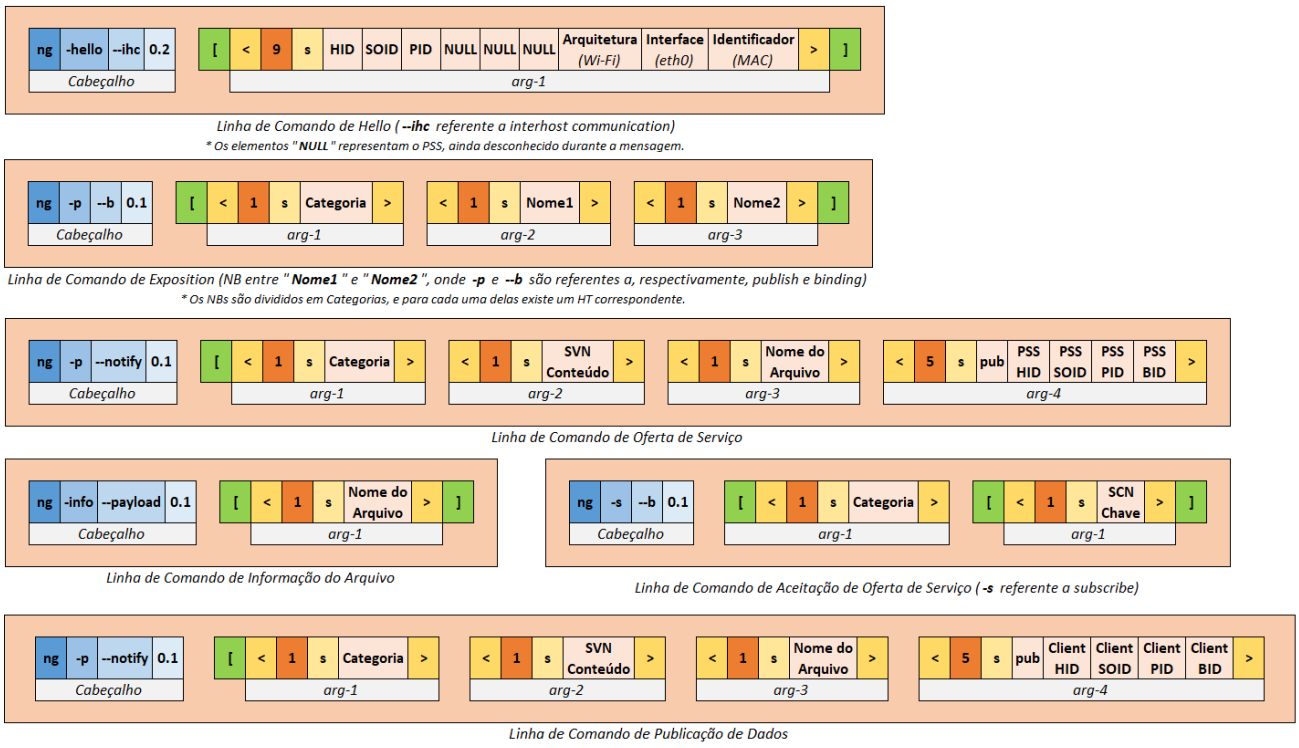

Figura 2.11: Linhas de Comando específicas da Mensagem NG embarcada.

Para identificar e entender melhor as Linhas de Comando particulares de cada tarefa presentes na Figura 2.11, é necessário conhecer o Ciclo de Vida NG, que pode ser definido pelos seguintes estados:

1. **Inicializar:** Preenchimento das informações e características do *Hardware* e geração dos SVNs;

2. **Descobrir:** Informação de sua presença aos pares, através das informações de seus respectivos SVNs e camada de enlace. É feito através da Mensagem de Descoberta (*Hello*), e possui a *Linha de Comando de Hello* como específica da tarefa;

3. **Expôr Recursos:** Expôr seus recursos ao par PGCS através do envio de NBs entre NLN e SVN ou entre SVNs. A Mensagem de Exposição das Palavras-Chave tem como Linha de Comando específica a *Linha de Comando de Exposição*;

4. **Oferecer Serviço:** Oferecer o serviço para as aplicações clientes interessadas no par PGCS. A Mensagem de Oferta de Serviço tem como Linhas de Comando específicas da tarefa a *Linha de Comando de Oferta de Serviço* e *Linha de Comando de Informação do Arquivo*;

5. **Aceitação de Serviço:** Recebimento do contrato com os SVNs da aplicação interessada, seguidos pela assinatura e envio do mesmo como finalização da negociação. A Mensagem NG de Aceitação de Oferta de Serviço possui as *Linhas de Comando da Aceitação de Serviço* como específicas para a tarefa;

6. **Publicar Dados:** Publicação dos dados para a aplicação cliente após o estabelecimento do contrato. O formato da Mensagem de Publicação de Dados NG tem como Linhas de Comando específicas a *Linha de Comando de Publicação dos Dados* e *Linha de Comando de Informação do Arquivo*, similar à tarefa de Oferta de Serviço;

7. **Finalizar:** Liberação de todas as estruturas alocadas.

NovaGenesis: Serviços de Publicação/Assinatura

Publish/Subscribe Service (PSS), em português, Serviço de Publicação/Assinatura, é um modelo de comunicação desenvolvido de modo que um serviço possa expôr seus NBs/-conteúdos ou descobrir outros NBs/conteúdos publicados. A permissão de acesso a essas informações é feita através de um contrato, onde o assinante segue as instruções do publicador, que por sua vez autentica e autoriza esta negociação [82].

Desta forma, o *Processo* PSS é composto pelo *Bloco* específico *Publish/Subscribe* (PS), em português, Publicação/Assinatura, e dos *Blocos* comuns HT e GW previamente descritos. O *Bloco* PS é responsável por realizar o encontro entre publicador e assinante; publicar/assinar os NBs e conteúdos; notificar os serviços envolvidos; e encaminhar o NB/conteúdo para um *Generic Indirection Resolution Service* (GIRS), em português, Serviço de Resolução de Indireção Genérico. O GIRS, através do *Bloco Binding Forwarding* (BF), em português, Encaminhamento de Ligação, seleciona um *Hash Table Service* (HTS), em português, Serviço de Tabela *Hash*, que através de sua estrutura de dados é capaz de armazenar ou entregar o NB/conteúdo [82].

NovaGenesis: Serviços de Rede

Os Serviços de Rede NG tem como objetivo integrar a arquitetura às tecnologias atuais, através do *Proxy/Gateway/Controller Service* (PGCS), em português, Serviço de *Proxy/Gateway*. Este serviço possui um *Bloco* específico *Proxy/Gateway* (PG) que realiza a função de

representar serviços e recursos de *hardware* físicos ou virtuais, além de encapsular o tráfego de dados NG sob uma tecnologia de rede física ou virtual e encaminhar mensagens para outros PGCSes [82].

Adicionalmente, o PGCS realiza operações de inicialização para serviços NG em um computador, sendo capaz de publicar a existência de PGCS, GIRS e HTS para o PGCS de diferentes *hosts* em um mesmo domínio. Por fim, ele também é responsável pela robustez da rede, enviando mensagens de Hello periodicamente via tecnologias de camada de rede, como Ethernet e Wi-Fi, permitindo aos PGCSes se auto-organizarem e manterem as propriedades da rede em caso de falha [82].

A fim de permitir o funcionamento da NG em sistemas embarcados, existe uma adaptação do PGCS, denominada *Embedded Proxy/Gateway Service* (EPGS), em português, Serviço de *Proxy/Gateway* Embarcado. Esta versão será utilizada neste trabalho para que a NG possa ser executada em um microcontrolador ESP32, que executará a função AS de um I4.0C. Dessa forma, o ESP32 será responsável pela comunicação entre um computador remoto com o *Processo* PGCS no qual se deseja obter dados da indústria, e os ativos industriais que serão conectados a um *Programmable Logic Computer* (PLC), em português, Computador Lógico Programável. O processo de comunicação entre computador remoto e ESP32 será feito via mensagens NG, enquanto a comunicação entre o ESP32 e o PLC será realizada via protocolo ModBus. A abordagem deste cenário será detalhada no Capítulo Cenário.

NovaGenesis: Serviços de Domínio

Domain Service (DS), em português, Serviço de Domínio, é um serviço para representar um domínio específico para o estabelecimento de hierarquias. Portanto, ele pode ser definido como um *Processo* que representa um domínio autônomo, estabelecendo contratos operacionais com outros domínios e encaminhando consultas de resolução de nomes para eles [82].

Como visto anteriormente, os PGCSes trocam mensagens "Hello"com outros PGCSes periodicamente, propagando ligações de nomes relacionados a outros serviços principais em um domínio. Como serviços de domínio e publicação/assinatura são anunciados para outros domínios, os DSes podem publicar ofertas *Service Level Agreement* (SLA), em português, Acordo de Nível de Serviço, para diferentes domínios, e em caso de acordo, estabelecerem um contrato expondo suas respectivas intenções para colaborar. Então, qualquer serviço de um domínio de mesmo nível pode solicitar a autorização do DS de outro domínio para publicar ou assinar NBs e conteúdos associados. Se autorizados, eles podem consultar NBs e dados públicos nos PSSes do domínio de mesmo nível. O acesso a NBs e conteúdos privados depende da autorização de outros serviços dentro do domínio da extremidade remota. Portanto, os SLAs podem indicar exatamente quais são os nomes acessíveis a outros domínios [82].

Como resumo, pode-se se dizer que a NG é a única abordagem que cobre a resolução de nomes entre domínios e *cache* de rede seguindo um modelo orientado a serviços.

NovaGenesis: Implementação

A fim de compreender melhor o funcionamento da NG, essa seção exemplifica passo-a-passo o processo de publicação e assinatura de um serviço qualquer, desde a inicialização da rede e fluxo de mensagens [82]. A Figura 2.12 ilustra a estrutura completa envolvida para o processo de publicação/assinatura.

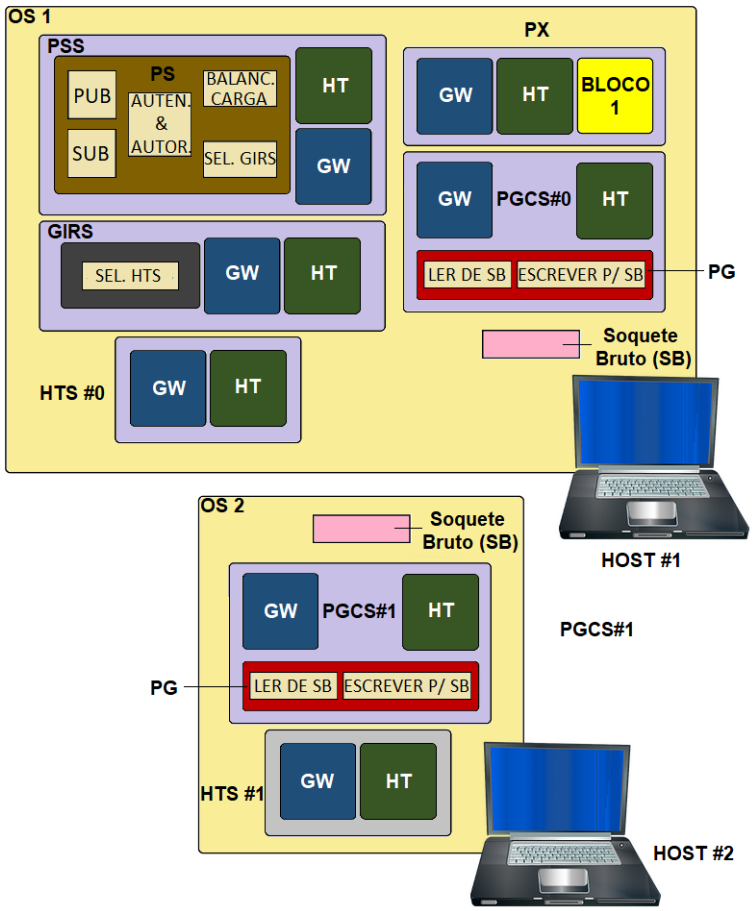

Figura 2.12: Implementação NG: Estrutura para comunicação NG entre diferentes *hosts*.

Conforme pode ser visto na Figura 2.12, o serviço a publicar o NB é o *Processo* denominado *PX*. Além disso, pode-se observar que a estrutura NG utiliza *Raw Sockets*, em português, Soquetes Brutos (SBs), que são um tipo de soquete de rede que permite que um aplicativo de *software* no computador envie e obtenha pacotes de informações da rede sem usar o sistema operacional do computador como intermediário.

Dada a estrutura para publicar/assinar NBs e/ou conteúdos representada pela Figura 2.12, o passo-a-passo para que o *Processo PX* publique um NB é descrito no fluxograma da Figura 2.13 a seguir:

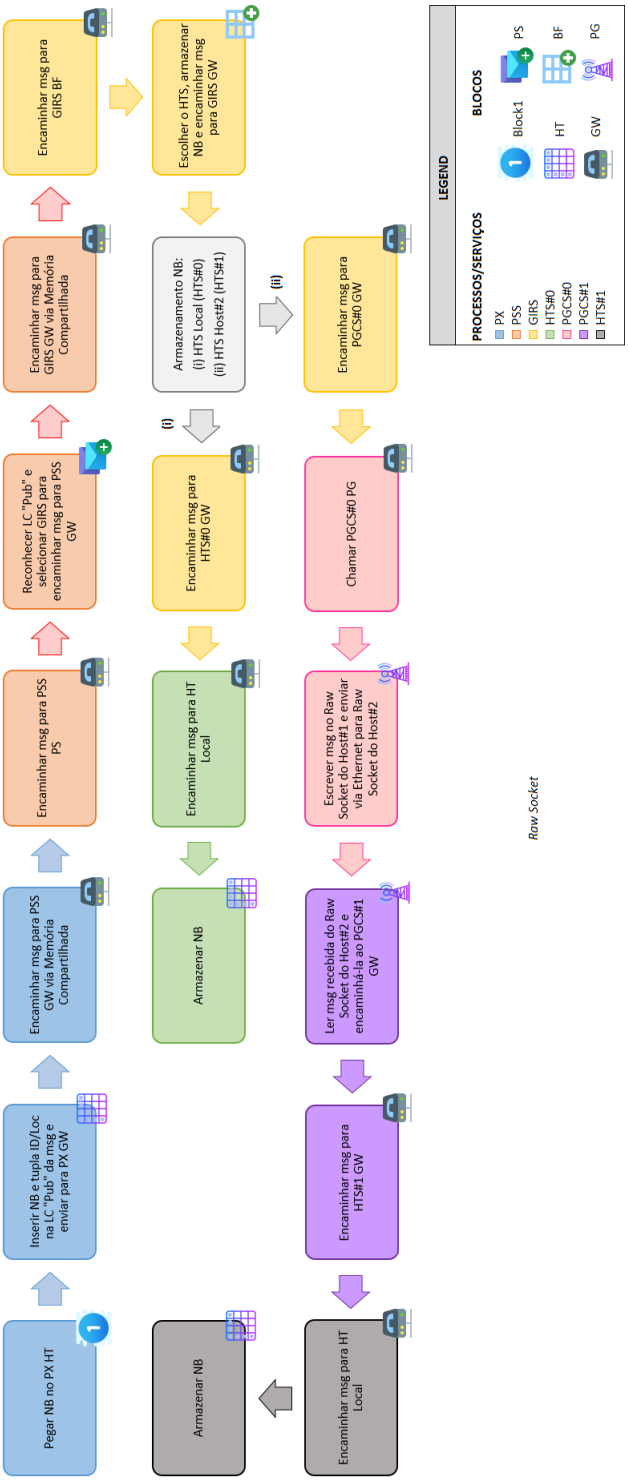

Figura 2.13: Implementação NG: Fluxograma para Publicação de NB.

Para a publicação de um NB, o primeiro passo é pegá-lo no *Bloco HT* do *Processo PX*, e inseri-lo na Linha de Comando de Publicação da mensagem juntamente com o seu identificador e localizador, para posteriormente enviá-lo ao *Gateway*. Através da memória compartilhada, a mensagem é encaminhada ao *Gateway* do *Serviço PSS* e enviada ao *Processo PS*, que por sua vez realiza o reconhecimento do tipo de mensagem (Pub) para selecionar o *GIRS*, cuja função é escolher o *HTS* para armazenar o NB, e enfim encaminhar a mensagem ao seu *Gateway*. Para o armazenamento local do NB, a mensagem é encaminhada ao *Gateway* de *HTS0*, responsável pelo *Bloco HT* Local, onde é enfim guardado. Já no caso de armazenamento em outro *host*, a mensagem é encaminhada ao *Gateway* de *PGCS0*, que chama o *Bloco PG* para escrevê-la no Soquete Bruto do *Host1* e enviá-la via Ethernet ao Soquete Bruto do *Host2*. No *Host2*, essa mensagem é recebida, lida e encaminhada do Soquete Bruto ao *Gateway* do *PGCS1*, onde é finalmente enviada ao *Gateway* de *HTS1* para ser armazenada no *Bloco HT* local.

Para a assinatura de um NB, o passo-a-passo é o mesmo, com a diferença de que a chave do NB é inserida na Linha de Comando *"Sub"* da mensagem, indicando uma abreviação ao termo *"Subscribe"*. Além disso, após a finalização dos passos ilustrados no fluxograma, o GW do HTS #1 entrega o NB para o GW do PGCS #1, que encaminha a mensagem para o *Bloco* PG escrever a mensagem no Soquete Bruto e transmitir a mensagem pela rede. Então, o *Bloco* PG do PGCS #0 lê o Soquete Bruto e recebe a mensagem, enviando-a para seu GW local, que por sua vez a encaminha para o GW do *Processo* PX, finalmente alcançando o Bloco 1 [82].

Capítulo 3

Trabalhos Relacionados

Neste capítulo será realizado um levantamento dos principais trabalhos relacionados ao tópico de Sistemas Ciber-Físicos Industriais. A revisão literária foi baseada na seleção de projetos relacionados que atendam as premissas essenciais necessárias em uma arquitetura ICPS baseada no modelo RAMI 4.0. Essas premissas incluem a representação virtual de ativos e a conexão dos mesmos à uma rede industrial com capacidade de armazenamento de dados em grande escala para monitoramento remoto, garantindo a integração dos mundos físico e virtual. As tecnologias que permitem que essa proposta se torne realidade foram definidas como "Dimensões", sendo divididas em D1-D4 e usadas para selecionar os trabalhos pesquisados que atendam a pelo menos uma das características. As Dimensões D1-D4 fazem parte das contribuições desta dissertação, sendo definidas hierarquicamente com o mesmo peso para esta análise, a partir de uma revisão literária de tecnologias imprescindíveis para o desenvolvimento de ICPSs e que também possam lidar com os problemas da Internet atual. A Tabela 3.1 ilustra as Dimensões D1-D4 para a revisão bibliográfica dos projetos de CPSs industriais, que incluem: IIoT, Desacoplamento ID/Loc com foco em identificadores únicos, Digitalização de ativos e *Cloud Computing* (CC).

Com esta revisão literária, o objetivo é apontar as lacunas dos trabalhos em relação: ao modelo de referência RAMI 4.0; à preocupação com as limitações da Internet atual no cenário da emersão de IIoT e dispositivos em massa conectados em rede; e à premissa de estruturas de nomeação capazes de aproximar o homem à máquina, cujo conceito é chave para o direcionamento inicial de uma arquitetura modelo para a Indústria 5.0.

A metodologia usada para determinar o estado-da-arte foi selecionar trabalhos que cobrem o maior número possível das Dimensões D1-D4, que são essenciais para cobrir os principais conceitos das camadas de Arquitetura do RAMI 4.0. Para isso, este Capítulo também apresenta uma tabela relacionando os trabalhos pesquisados com o modelo de referência de arquitetura RAMI 4.0 descrito no Capítulo anterior, além de apontar lacunas existentes quanto ao suporte às limitações da Internet atual.

Em [104], Liu e Su implementaram um modelo de virtualização de recursos de manufatura de tal forma que eles sejam vistos como serviços, apontando a SOA como uma característica

Tabela 3.1: Dimensões D1-D4 para o estado-da-arte dos projetos de ICPSs pesquisados.

Dimensão	**Tecnologia**	**Descrição para o Cenário I4.0**
D1	IIoT [93], [94], [95]	Infraestrutura que permite que os ICPSs sejam conectados em rede. É uma rede de ativos industriais que podem ser conectados em larga escala e que se comunica via Internet e troca dados em tempo real. Para esta dimensão, é necessário se atentar à limitação da Internet atual da quantidade de endereços IP disponíveis, dada a possibilidade de um número significante de dispositivos em uma rede IIoT.
D2	Desacoplamento ID/Loc [96], [97]	Proposta adotada no cenário IIoT que consiste em unicamente identificar e localizar ativos conectados na rede de forma eficiente e única, solucionando problemas relacionados a dispositivos com mesmo identificador e localizador na rede.
D3	Digitalização [98], [99], [100]	Abstração digital de uma entidade física. Usado para representar digitalmente os ativos, funcionalidades e características, permitindo seu gerenciamento e monitoramento, sendo a Dimensão essencial do modelo RAMI 4.0
D4	Cloud Computing [101], [102], [103]	Modelo usado para armazenamento de dados e processamento, suportando um sistema industrial mais robusto considerando ativos industriais e dados em larga escala.

importante para a Indústria 4.0. Como resultado, o projeto compreende a digitalização da manufatura e detalhamento do seu processo de implantação, que são características pontuais para o desenvolvimento de Fábricas Inteligentes. Este trabalho foca em D3 para a digitalização de ativos, mas não expande à possibilidade de representação de dispositivos em massa e divisão entre ID/LOC dos mesmos.

Karakostas desenvolveu em [105] uma arquitetura DNS capaz de traduzir IDs únicos de objetos físicos em endereços de objetos na rede e coletar suas informações, como por exemplo localização. Este projeto foi proposto em uma infraestrutura de logística de transporte, e sua abordagem apresenta ideias de suporte para Desacoplamento ID/Loc, sendo focado na Dimensão D2, e portanto a digitalização de ativos não é abordada.

Os autores Jazdi e Nasser [106] criaram em Stuttgart uma extensão CPS em um sistema embarcado usando o microcontrolador Cerebot 23MX7 como *gateway*. Este *gateway* se comunica com a nuvem via protocolo HTTP e com uma máquina de café via comunicação *Controller Area Network* (CAN). Foi criado um aplicativo Android e um site para a operação do CPS capaz de trocar dados com a máquina acessando a nuvem via *smartphone*. Muitos benefícios foram percebidos por este experimento, como a capacidade de identificar produtos defeituosos ou esgotados na máquina de café. D3 e D4 são usados em conjunto para a digitalização fabril com o recurso de CC, mas limitando o identificador ao seu localizador, já que não é realizada uma abordagem sobre o conceito de desacoplamento.

Ungurean *et al.* propôs em [107] uma arquitetura baseada em especificações OPC.Net que permite a participação dos clientes nos processos industriais. Um módulo servidor de dados criado é responsável por receber dados da rede de sensores e enviar comandos para atuadores via comunicação sem fio. Já o aplicativo HMI, executado em computadores, *tablets* e *smartphones*, é utilizado para que os clientes leiam informações da rede de sensores ou enviem comandos a serem executados em atuadores. Para a implementação, foi criado um *gateway* para aquisição de dados em tempo real usando um microcontrolador ARM Cortex X3. Um dos benefícios observados foi o fato de que a integração de novos barramentos de campo no servidor OPC.Net não exigia atualização ou recompilação de toda a aplicação. São usados métodos que abordam conceitos de IIoT em D1, mas não é realizada uma aplicação prática com digitalização de dispositivos industriais.

Em [108], Yu *et al.* propuseram uma arquitetura CPS para detecção de equipamentos de energia baseado em I4.0. Dentre as tecnologias usadas, destaca-se instrumentação virtual, detecção e medição, integração mecânica e elétrica, comunicação por rede e cliente móvel. Devido à otimização geral do processo de detecção, o sistema pode alterar rapidamente seus padrões e ajustar as funções de detecção, atendendo as necessidades de 19 tipos de teste de dispositivos elétricos e inspeção de qualidade, com as vantagens de eficiência, flexibilidade e segurança. Comparado ao método convencional e à tecnologia de detecção flexível, o teste médio economiza de 2 a 3 pessoas e a eficiência da detecção é aprimorada em pelo menos 300%. O foco está em D2, onde o desacoplamento ID/LOC é utilizado para detectar equipamentos, mas não é feita uma abordagem sobre como representá-los digitalmente.

Uma abordagem foi proposta em [109] por Grangel-González *et al.* para representar semanticamente as informações de dispositivos I4.0 com um AS baseado em *Resource Description Framework* (RDF), usando um esquema de identificação de recursos uniforme para identificar todos os tipos de entidades relevantes, como objetos físicos, suas propriedades, conceitos abstratos e dados derivados. O acesso unificado às informações foi feito por meio da linguagem de consulta RDF chamada SPARQL. Como exemplo, um AS semântico de um controlador servo motor foi gerado. Os autores pretendem expandir o vocabulário AS no futuro para uma gama mais ampla de dispositivos. Este projeto envolve D2 e D3 para representação digital e desacoplamento ID/LOC, porém trata-se de um trabalho conceitual, onde não é feita uma aplicação prática de uma rede com dispositivos físicos sendo digitalizados.

Em [110], foi proposto um projeto europeu voltado para a otimização e fabricação de processos de ativos denominado *Cloud Collaborative Manufacturing Networks* (C2NET), em português, Redes de Manufatura Colaborativas em Nuvem. Seu principal objetivo é construir ferramentas habilitadas para nuvem visando otimizar a cadeia de suprimentos de ativos de manufatura e logística de pequeno e médio porte com base em planos de demanda, entrega e produção colaborativos. O C2NET é uma arquitetura em camadas com infraestrutura em nuvem, baseada nos conceitos: *Everything as a Service* (XaaS), em português, Tudo como um Serviço; *Infrastructure as a Service* (IaaS), em português, Infraestrutura como um Serviço; *Platform as a Service* (PaaS), em português, Plataforma como um Serviço; e *Software as a Service* (SaaS), em português, *Software* como um Serviço. Na camada SaaS, as principais funções existentes são: coleta de dados de empresas usando o paradigma Pub/Sub; análise da coleta de dados e ajustes dinâmicos; e aplicativo para interface do usuário industrial, exibindo dados em tempo real para o monitoramento, controle, compartilhamento e colaboração das partes interessadas na rede de fornecedores. Na camada PaaS, os serviços são divididos em: PaaS para Fabricação, com serviços baseados em nuvem comumente usados; e PaaS para SLA usado para ativos de alto nível afim de garantir QoS. Por fim, a camada IaaS oferece computadores (máquinas físicas ou virtuais) e outros recursos sob demanda por meio de grandes *pools* de equipamentos instalados nos *data centers*. O projeto foi testado nos setores automotivo, metalúrgico e dermocosmético, comprovando sua eficiência. Apesar deste trabalho cobrir todas as Dimensões, considera-se o uso da estrutura da Internet atual para sua implementação, ou seja, para um futuro promissor onde haja uma nova arquitetura evolutiva, o modelo ICPS deve ser redesenhado.

Pisching *et al.* elaborou em [111] uma proposta de arquitetura CPS baseada no modelo 5C a fim de melhorar a arquitetura citada para introduzir os conceitos de IoT e IoS. O projeto foi testado em uma bancada dividida em 5 estações com módulos de produção de peças: distribuição, teste, manuseio, processamento e classificação das peças. Cada módulo é composto por sensores e atuadores, um PLC para controlá-los, um computador na rede local para leitura de variáveis através do servidor OPC Festo Inc. (Codesys V2.3) e uma biblioteca do cliente OPC para C# disponível no site da OPC Foundation. Os computadores locais são responsáveis pela leitura e análise dos dados da estação, além de permitir a comunicação com os computadores locais das demais estações e com o computador central, responsável por coletar as informações dos objetos inteligentes utilizando o protocolo TCP/IP e criar o modelo virtual do sistema. Este modelo virtual é desenvolvido em linguagem C# orientada a objetos para analisar as condições dos objetos inteligentes, como falha de atuação, tempo de uso e informar aos usuários sobre as necessidades de manutenção. Este trabalho foca em D1 para o desenvolvimento de uma rede de dispositivos conectados em massa, mas não detalha como digitalizá-los ou suportar as demais limitações da Internet atual.

Jiang se baseou no modelo da Arquitetura 5C com o propósito de expandi-la em [112] para uma Arquitetura 8C ao adicionar diretrizes de Aliança, Cliente e Conteúdo. A diretriz Aliança é responsável por integrar a cadeia de valor e a cadeia de produção entre as diferentes partes dos processos de produção, que podem construir em conjunto uma cadeia de suprimentos e reprogramar a linha de produção para formar uma cadeia de produção em tempo real. A diretriz Cliente foca na sua participação desse no processo produtivo, no qual ele é capaz de especificar detalhes do *design* do produto ou até mesmo modificar produtos durante o processo, dando origem a serviços de melhoria de QoS. Por fim, a diretriz Conteúdo permite a extração, armazenamento e rastreabilidade do produto, no qual todas as informações de produção, como fontes de matéria-prima e fornecedores, processos, fenômenos ambientais e parâmetros de produção, são extraídas e armazenadas em um banco de dados para análises futuras. O projeto é benéfico para o desenvolvimento de Fábricas Inteligentes. Similar ao trabalho descrito anteriormente, o projeto não cobre a digitalização de ativos industriais.

Yun *et al.* desenvolveram [113], uma nova arquitetura de plataforma de gêmeos digitais de grande escala chamada *universal Digital Twin* (uDiT), em português, Gêmeos Digitais universal. A comunicação é centrada em dados flexíveis baseada em *Object Management Group* (OMG) DDS e funções de co-simulação baseadas em *Functional Mockup Interface* (FMI). A plataforma tem como proposta abstrair dispositivos através da representação de sua entidade virtual. Ela também suporta o compartilhamento de dados pelas entidades em qualquer formato, além de funções de interoperação para um grande número de gêmeos digitais e sistemas físicos. Este projeto tem como foco o uso de gêmeos digitais em larga escala, enfatizando D1 e D3, mas não faz uso de desacoplamento ID/LOC para garantir ativos com identificadores únicos, além de não suportar CC para melhor organização de dados monitorados em massa.

Em [114], Ayatollahi *et al.* implementaram uma SOA em produtos de manufatura inteligentes baseados em RAMI 4.0 para identificação, acesso a dados e controle. A proposta foi implementada em uma torre de ferramentas utilizada como suporte em fabricação para fixar

ou soltar peças para montagem e realizada pelo manuseio de um robô. Para tanto, serviços foram desenvolvidos pelo servidor OPC UA, como descoberta, leitura, link de partes interessadas e criação de assinaturas, utilizando uma versão de teste de um *Software Development Kit* (SDK), em português, Kit de Desenvolvimento de *Software* (SDK), C++ fornecido pela UnifiedAutomation GmbH. Foi implementada uma modelagem de informação OPC UA para clientes lerem os objetos e executarem comandos através do padrão MTConnect, que oferece um vocabulário semântico para a comunicação entre máquinas, utilizando a ferramenta UA Modeler. Essa ferramenta gera fragmentos dos códigos apropriados ao SDK, que precisam ser incluídos no ambiente de programação, facilitando o desenvolvimento da aplicação no servidor OPC UA. Em seguida, o projeto é compilado com CMake em uma plataforma baseada em Linux e o arquivo transferido para um Raspberry Pi para controle de sensor e atuador. Com um cliente OPC UA genérico denominado UAExpert, o aplicativo de servidor OPC UA desenvolvido foi testado e foi observado que com o conjunto de dados e funções implementadas, os dispositivos de fábrica devem ser capazes de descrever a si próprios, seus status e oferecer suas funcionalidades. Conceitos primordiais do modelo RAMI 4.0 são usados, como D3 para digitalização dos dispositivos com capacidade de auto descrição e funcionalidades, mas não considera possíveis problemas advindos das limitações da Internet atual, como a proliferação de dispositivos em massa e o desacoplamento ID/LOC.

Paulo *et al.* propôs em [115] o projeto NodeI4.0, que consiste em um protótipo embarcado capaz de adicionar o Nível Conexão Inteligente da Arquitetura 5C em qualquer sistema legado. O *hardware* possui circuitos de entradas e saídas digitais analógicas, além de um módulo ESP8266-01 para conexão à Internet e um microcontrolador PIC18F2550 para a transmissão de variáveis *Input/Output* (I/O), em português, Entrada/Saída, da comunicação serial. O NodeI4.0 tem 2 modos de operação: Ponto de Acesso, usado para inserir as configurações empregadas em outro modo, denominado Estação. No modo Estação, o projeto se conecta a um roteador e envia os dados de I/O a um servidor *broker* via Wi-Fi. Reiniciando o NodeI4.0, uma rede Wi-Fi é gerada com o *Service Set Identifier* (SSID), em português, Identificador do Conjunto de Serviço, chamado "NodeI4.0". Em seguida, conectando-se à rede Wi-Fi através de uma interface *Dynamic Host Configuration Protocol* (DHCP) e inserindo a senha correta, é possível acessar a página de configuração. A página permite que o usuário ative 3 sub-modos: "ler", para ler as entradas analógicas do microcontrolador; "Escrever", para escrever as saídas digitais; e "Leitura/Escrita"para a combinação de ambos os modos. Ao ativar um dos sub-modos, é possível enviar dados ao servidor através de um *Uniform Resource Locator* (URL), em português, Localizador de Recursos Uniforme, via protocolo HTTP de solicitação. O protótipo resultou em eficiência para sistemas cujo requisito de resposta é superior a 50 milissegundos (ms). É realizado um trabalho em D3 para monitoramento de dispositivos, porém limita-se o identificador ao localizador, além do suporte da arquitetura para casos onde haja uma rede com um número de dispositivos conectados em grande escala.

Os autores Xiaoqing *et al* implementaram em [116] uma SOA escalável chamada *Cyber-Physical Manufacturing Cloud* (CPMC), em português, Nuvem de Manufatura Ciber-Física, foi desenvolvida, na qual cada fabricante em uma indústria possui máquinas monitoradas por controladores que se comunicam com seu próprio servidor local via protocolos TCP/IP

e MTConnect. Os controladores também podem enviar dados para a nuvem via protocolo HTTP, que é responsável pela comunicação entre a nuvem e os consumidores. A arquitetura é dividida em cinco camadas: A Camada de Recursos contém os recursos de manufatura mantidos pelos fabricantes que fornecem serviços em nuvem em um modelo Pub/Sub. Esses recursos de manufatura se comunicam com os servidores da Camada de Virtualização de Recursos via TCP/IP para enviar solicitações de operações e receber resultados; e MTConnect para monitorar e coletar dados de recursos. A Camada de Virtualização de Recursos usa métodos de publicação de serviços da *Web Representative State Transfer* (RESTful) para virtualizá-los. Esses serviços da web são hospedados e executados nos servidores locais dos fabricantes, protegendo os recursos por meio de solicitações de acesso à Internet autenticadas. Essa camada se comunica por meio do protocolo REST baseado em HTTP com a Camada Núcleo da Nuvem, responsável por hospedar serviços de nuvem básicos, como serviços de assinatura de usuário, gerenciador de segurança, gerenciador de repositório de serviços de virtualização e interface de programação de aplicativos públicos, privados, comerciais e não comerciais. Por fim, a Camada Núcleo da Nuvem se comunica por meio do protocolo REST baseado em HTTP com a Camada de Aplicação, que gerencia aplicativos para os clientes realizarem operações de fabricação na Internet em plataformas como o navegador da Web, sistemas incorporados, *desktop* e sistemas operacionais móveis. A arquitetura proposta foi testada com um protótipo operacional em grande escala e comprovada viabilidade para monitorar e executar operações em nuvem e processos de manufatura pela Internet. Este trabalho cobre todos as Dimensões descritas, mas considera essencial a estrutura de Internet atual para sua aplicação, limitando a uma rede com baixa flexibilidade e suporte a diferentes protocolos.

Em [117], um projeto europeu chamado *Cloud-Based Rapid Elastic Manufacturing* (CREMA), em português, Manufatura Elástica e Rápida baseada em Nuvem, foi desenvolvido. Seus principais princípios são a Virtualização da Manufatura, a Interoperabilidade, a Otimização, a Colaboração, os Processos de Manufatura em Nuvem e a Integração das Partes Interessadas. Para os princípios de Virtualização e Interoperabilidade da produção, é desenvolvido um conjunto de componentes que oferece: suporte à coleção de modelos semânticos relacionados à produção definida em RDF ou *Ontology Web Language 2* (OWL2); suporte à transformação e integração dos dados existentes; suporte a diferentes tipos de banco de dados, como dados estruturados, semânticos e binários para gerenciamento da plataforma; acesso unificado às fontes de dados do sensor relacionadas à produção e CPS; permissão do uso de objetos inteligentes, nós de sensores e redes de sensores sem fio; paradigma Pub/Sub; e virtualização do serviço, representando ativos e serviços de manufatura do mundo físico. Para os princípios do Processo de Manufatura em Nuvem e Otimização, os componentes são responsáveis por: permitir a elaboração do processo produtivo com base nos ativos fabris disponíveis; execução dos processos definidos por planos de serviço; instanciar serviços sob demanda em recursos da nuvem, monitorando sua disponibilidade; otimizar modelos de processos baseados em serviços. Finalmente, os princípios de Colaboração e Integração das Partes Interessadas são compostos por componentes com funções para: monitorar as instâncias de processos em execução; fornecer funcionalidades de análise de negócios para obter informações significativas por meio de dados de sensores e processos no domínio da manufatura; oferecer suporte aos

trabalhadores de campo por meio de um aplicativo para *tablet* que permite aos supervisores instruir os trabalhadores que usam óculos inteligentes; e englobar todos os componentes que interagem com os usuários, de forma a incluir informações sobre os processos de produção, alertas e problemas de execução. Os resultados foram direcionados para que as orquestrações de manufatura entre as organizações e a integração de recursos distribuídos fossem permitidas, tornando os processos de manufatura mais eficientes. Similar ao trabalho anterior, apesar de cobrir todas as Dimensões, é voltado unicamente ao modelo de Internet e suas limitações, sendo necessária uma mudança em toda sua estrutura no caso de uso em uma arquitetura evolutiva diferente da vigente.

Em [118], Kannengiesser *et al.* propuseram o SITCHEN 4.0, que é um método de engenharia baseado em pontos de vista de diferentes níveis para modelagem CPS baseado na padronização RAMI 4.0 e semântica de rede. O projeto descreve os seguintes níveis de abstração: (i) Nível de Instância, contendo o CPS incorporado nos mundos físico e virtual; (ii) Nível do Modelo, contendo tipos de CPS representados por meio de visualizações; (iii) Nível de Meta-modelo, contendo os pontos de vista, responsável por especificar as convenções (notações, linguagens e tipos de modelo) para a construção de um tipo de vista; e (iv) Nível Meta-Metamodelo, definindo os padrões semânticos de RDF e OWL para construir e representar pontos de vista. Baseado nas 3 dimensões do RAMI 4.0, a modelagem do Ponto de Vista é escolhida de acordo com o aplicativo CPS selecionado, como por exemplo Controle de Produção e Processo de Negócios. Consequentemente, cada ponto de vista está associado a diferentes tipos de modelos, como por exemplo Objeto de Dados e *Standard Query Language* (SQL) para Processo de Negócios. Em seguida, os participantes podem criar seus próprios modelos de visualizações e CPS com base nos tipos de modelo de Ponto de Vista, que são definidos como classes e modelados em RDF ou OWL. Exemplos foram mostrados, como a criação dos atributos *Produto*, *Nome do cliente* e *Data de Entrega* para o modelo SQL do ponto de vista de Controle de Produção. Finalmente, com o conjunto de visualizações desenvolvido, os modelos podem ser transformados em instâncias CPS, que por sua vez têm sido utilizadas como a principal aplicação que gera código executável dinamicamente ou sob demanda (por exemplo, em PLCs) com interfaces padrão da I4.0, como OPC UA, *Message Queuing Telemetry Transport* (MQTT), *Advanced Message Queuing Protocol* (AMQP) e *PLCopen extensible Markup Language* (XML). Focado em D3, este projeto é direcionado à representação de dispositivos industriais, mas não há uma aplicação real com uma rede IIoT e CC, limitando à digitalização de apenas um único ativo, além do identificador e localizador serem idênticos para a aplicação.

Junior *et al.* propuseram em [119] uma arquitetura CPS baseada na Arquitetura 5C, composta por 5 módulos. O Módulo de Configuração é responsável pela interface de configuração de todos os módulos e consiste em um *setup* e um modelo topológico. O Módulo de Inteligência mapeia e rastreia padrões, comportamentos e controle de qualidade de dados, usando *Predictive Model Markup Language* (PMML). O Módulo Cibernético gerencia as informações do sistema em escala de tempo, usando o protocolo OPC UA via linguagem XML e um *buffer* de dados. Um Módulo de Conversão converte os dados coletados no Módulo de Comunicação em informações para o sistema por meio da atribuição de semântica e possui

um Agente MTConnect. Por fim, o Módulo de Comunicação é responsável por adaptar os diferentes protocolos de redes industriais ao padrão adotado pelo sistema, garantindo sua interoperabilidade, e possui um Adaptador MTConnect. Para os testes, foi empregada uma planta industrial de ensino Smart PD3, composta por equipamentos como transmissores de temperatura, vazão e nível, e um PLC transmitindo seus dados via ModBus TCP/IP. Assim, um Adaptador ModBus TCP/IP utilizando uma linguagem de programação C++ foi implementado para transformar os dados obtidos da planta no padrão XML MTConnect, bem como sua modelagem topológica. Além disso, um Agente C++ MTConnect foi criado para publicar os dados do Adaptador em uma página padrão HTTP. O Agente e o Adaptador são executados em um Raspberry Pi 3 conectado à rede Ethernet do PLC. Por fim, um computador foi utilizado para acessar a página que fornece o XML gerado pelo MTConnect, contendo eventos como: amostras e condições dos equipamentos em funcionamento na planta. Os resultados mostraram precisão dos dados. Para a aquisição de dados de ativos, é usado D3, mas não é implementada a conexão prática de dispositivos em rede com suporte ao uso de IIoT e CC.

Em [120], Pisching projetou uma arquitetura para descoberta de equipamentos em processos de manufatura com foco em I4.0 com base no modelo de referência RAMI 4.0. Ele é capaz de fornecer componentes para permitir a comunicação entre produtos e equipamentos. Também é oferecido um serviço web capaz de fornecer um mecanismo semelhante ao DNS para localizar equipamentos a serem processados. Baseado no modelo RAMI 4.0, a Camada Ativo contém os elementos físicos, que incluem um identificador único e um AS integrado com dispositivos de controle. A Camada de Integração consiste em um *driver* para gerenciamento de operações de fabricação e um servidor atualiza os dados e interage com os dispositivos de controle. Para a Camada de Comunicação, que permite a troca de informações entre os serviços da camada superior, a tecnologia Ethernet é considerada como meio físico e o TCP/IP para a comunicação de serviços virtuais. A Camada de Informação, que consiste em dados de produtos e equipamentos, é formada por entidades virtuais e uma rede hierárquica de equipamentos. Na Camada Funcional, o componente de serviço da web para gerenciamento de operações de fabricação determina o equipamento que processa o produto e atualiza a operação concluída. Finalmente, a Camada de Negócios inclui recursos baseados em SOA implementados como serviços da web para fornecer soluções relacionadas aos processos de negócios. A modelagem funcional e conceitual da arquitetura foi desenvolvida utilizando a técnica de *Production Flow Schema/Petri Network* (PFS/PN), em português, Esquema de Fluxo de Produção/Rede de Petri, e aplicada em um sistema de produção modular, mostrando sua eficiência. Este trabalho atende à D2, D3 e D4, sendo uma alternativa ao modelo RAMI 4.0, mas não considera o suporte a uma rede IIoT com dispositivos em massa conectados.

Antes de correlacionar esses trabalhos e determinar as oportunidades de pesquisa, traz-se na Figura 3.1 uma linha evolutiva desses trabalhos relacionados, incluindo os modelos de referência de arquitetura. Isso permite que os mesmos sejam ponderados pelo ano de origem, dando uma visão de como os modelos de arquitetura de referência evoluíram ao longo do tempo.

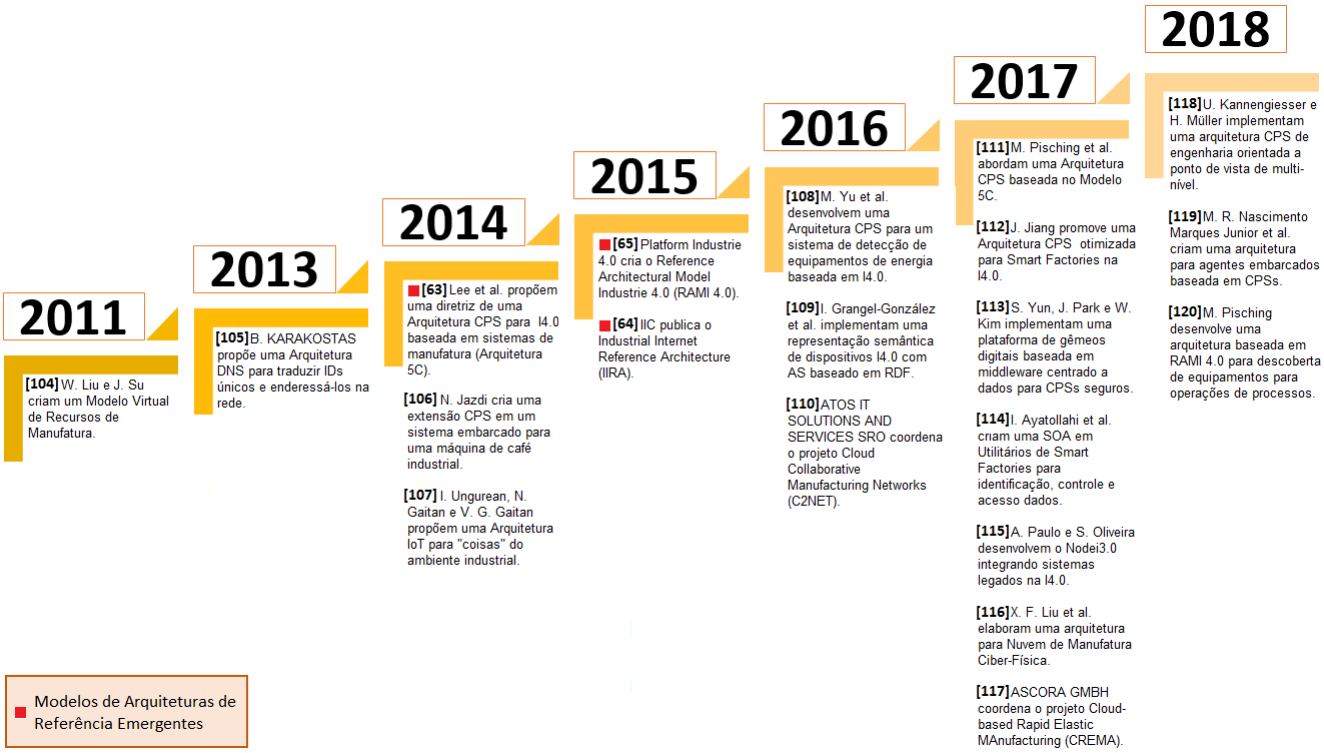

Figura 3.1: Linha Evolutiva dos trabalhos relacionados e modelos de arquitetura de referência.

Como pode ser visto na Figura 3.1, já haviam propostas relacionadas à virtualização de manufatura e dispositivos unicamente identificáveis para o contexto I4.0 antes do surgimento dos modelos de arquitetura de referência de ICPSs. Apesar disso, esses projetos também contêm ideias e semelhanças com as propostas divulgadas por esses modelos. Em outras palavras, os conceitos-chave dos modelos de referência de arquitetura são utilizados no ambiente industrial há muito tempo, mas como pode ser visto na linha evolutiva, o surgimento desses modelos resultou em um aumento significativo nas propostas de ICPSs, comprovando sua eficácia e facilidade de implementação no cenário I4.0.

Baseado na pesquisa de projetos relacionados a ICPSs, a Tabela 3.2 realiza um breve resumo dos trabalhos através de sua descrição, relação com o modelos de arquitetura de referência RAMI 4.0 previamente descrito e principais tecnologias I4.0 usadas.

Tabela 3.2: Correlação entre projetos de ICPSs, Dimensões D1-D4 para I4.0 e arquiteturas de referência.

Ref.	Objetivo Principal	D1 - IIoT	D2 - ID/Loc.	D3 - Digital.	D4 - CC	Relação com RAMI 4.0
[104]	Proposta de arquitetura de recursos de manufatura dinâmica.			x		Representação virtual de recursos e atendimento aos requisitos de digitalização no ciclo de vida do produto.
[105]	Arquitetura baseada em DNS para converter IDs únicos de objetos físicos em endereços de objetos.		x			Identificadores únicos e desacoplamento de ID/Loc exigido pelo AS e I4.0C.
[106]	Aplicação CPS para uma máquina de café industrial.			x	x	Virtualização da máquina e monitoramento remoto.
[107]	Apresenta uma arquitetura baseada em OPC.NET para setores industriais e de prédios inteligentes.	x				Contextualização de dados e método de comunicação que pode ser usado para integrar diferentes setores industriais.
[108]	Arquitetura CPS para detecção de equipamentos baseada em I4.0.		x			Contextualização de dados e integração da Manuf. pela conexão de processos.
[109]	Representação semântica de dispositivos I4.0 com AS baseado em RDF.		x	x		Representação digital e IDs de dispositivos baseados em semântica e entidade virtual.
[110]	Redes de Manuf. Colaborativas em Nuvem (C2NET).	x	x	x	x	Módulos C2NET cobrem as principais premissas das Camadas RAMI 4.0 e integra toda manufatura.
[111]	Arquitetura 5C otimizada para I4.0.	x				Objetos inteligentes capaz de se comunicarem com todos o setores fabris.
[112]	Arquitetura 5C adaptada para I4.0.	x				Integração das cadeias de valor e produção entre processos de manufatura.
[113]	A novel architecture for large-scale digital twins (uDiT).	x		x		Camada de Integração: conceito de gêmeo digital.
[114]	SOA em utilitários de manufatura inteligentes para identificação, acesso de dados e controle.			x		Capacidade de auto-descrição dos dispositivos, como seu status e funcionalidades, e coleta de dados que podem ser contextualizados em informações.
[115]	NodeI4.0: adição do Nível Conexão Inteligente da Arq. 5C para sistemas legados.			x		Coleta de dados de máquinas e dispositivos para monitoramento remoto.
[116]	Nuvem de Manufatura Ciber-Física (CPMC) integrando nuvem, CPSs and manuf.	x	x	x	x	Coleta de dados, virtualização, CC, contextualização de dados, operações para cada ativo e serviços de aplicativo.
[117]	Manuf. Elástica e Rápida baseada em Nuvem (CREMA) para agilidade e escalabilidade.	x	x	x	x	O projeto contém todas as premissas relevantes incluídas nas Camadas RAMI 4.0.
[118]	Método de engenharia de ponto de vista de multi-níveis para CPS.			x		Representação dos ativos físicos no mundo virtual.
[119]	Arquitetura 5C para ambiente industrial.			x		Aquisição de dados de ativos, processamento e descrição de funções.
[120]	Arquitetura para descoberta de equipamentos em processos de manufatura para I4.0.		x	x	x	Premissas do RAMI 4.0 atendidas com foco no setor de manufatura.

Como pode ser visto nos trabalhos relacionados, os projetos de ICPSs utilizam o cenário de Internet atual para a virtualização de ativos e coleta, monitoramento e controle de dados, sendo muitas vezes complexa a introdução de tecnologias emergentes a este tipo de arquitetura. Baseado na Tabela 3.2, pode-se notar que D3 é a dimensão mais utilizada, devido à sua importância no modelo RAMI 4.0 para a digitalização de ativos. O suporte à IIoT e CC não foram explorados, de modo que os protótipos apresentados não consideram uma aplicação com dispositivos em massa conectados em rede e aquisição de seus dados. Ainda em relação à essa questão, a limitação de endereços IP não é abordada, tornando possíveis problemas futuros para este cenário, já que estes trabalhos se baseiam na arquitetura da Internet atual. A questão de desacoplamento ID/LOC também é abordada de forma tímida, desconsiderando casos onde há mudança de localização de ativos e interferência em seu identificador, que seguindo o modelo de referência deve ser único. Desta forma, uma arquitetura FI baseada no RAMI 4.0 seria vantajoso, pois além de apresentar as principais dimensões de um ICPS, também visam o futuro industrial ao se preocupar com as limitações da Internet atual no cenário I4.0.

Visto que FI aplicado ao contexto de ICPSs é uma novidade a ser explorada no cenário fabril devido à grande parte dos projetos atuais usarem uma rede de dispositivos industriais legada, o Capítulo a seguir apresenta com detalhes uma proposta de arquitetura FI para Sistemas Ciber-Físicos Industriais com suporte às tecnologias emergentes de I4.0, baseando-se no modelo de referência RAMI 4.0. Arquiteturas de FI atacam diferentes problemas relacionados às limitações da Internet atual, que também devem ser considerados para os ambientes industriais devido à sua emersão no cenário I4.0 juntamente com IIoT. A utilização de FI em ICPS visa mostrar que é possível introduzir uma arquitetura inovadora baseada no modelo RAMI 4.0 que suporte todas as dimensões descritas anteriormente e também se atente aos problemas da Internet atual destacados nas seções anteriores, como por exemplo: limitação de endereços IP para ativos em massa conectados na rede, identificação única de dispositivos industriais, desacoplamento ID/LOC, flexibilidade e heterogeneidade da rede. Como diferencial, a arquitetura FI NG que será usada neste trabalho é direcionada ao futuro industrial devido à sua característica centrada a nomes, com uma estrutura de nomeação que permite ao usuário utilizar linguagem natural de forma a aproximar o homem à máquina, sendo um conceito primário de Indústria 5.0 para um modelo de arquitetura inicial e inovador.

Capítulo 4

Cenário de Testes para Monitoramento Remoto de Ativos Industriais

Neste capítulo é apresentado o cenário integrado desenvolvido como principal contribuição dessa dissertação. Ele visa a adoção da NG como Sistema Ciber-Físico em Indústria 4.0, seguindo o modelo RAMI 4.0. Primeiramente, é apresentado o cenário geral do projeto através de seu diagrama completo para descrever toda a proposta desenvolvida, além dos equipamentos de *hardware* e o *software* utilizado para implantação real. Em seguida, para melhor entendimento e organização do capítulo, o cenário será detalhado em 2 partes, sendo: a Parte 1 responsável pela implementação de uma réplica do cenário industrial existente nos ambientes fabris, com ativos, protocolos e comunicação legados comumente usados, e o monitoramento dos dispositivos industriais sendo realizados por um controlador lógico programável atráves de um *buffer*; e a Parte 2 com as novidades apresentadas por esta dissertação, através de microcontroladores para a aquisição dos dados dos dispositivos industriais no ambiente fabril armazenados no *buffer* de monitoramento do controlador e a digitalização de ativos através da FI NG, criando um ICPS baseado no modelo RAMI 4.0.

4.1 Cenário Geral

Para compreender melhor o cenário geral do projeto, a Figura 4.1 ilustra o diagrama completo da aplicação da NG como CPS em I4.0.

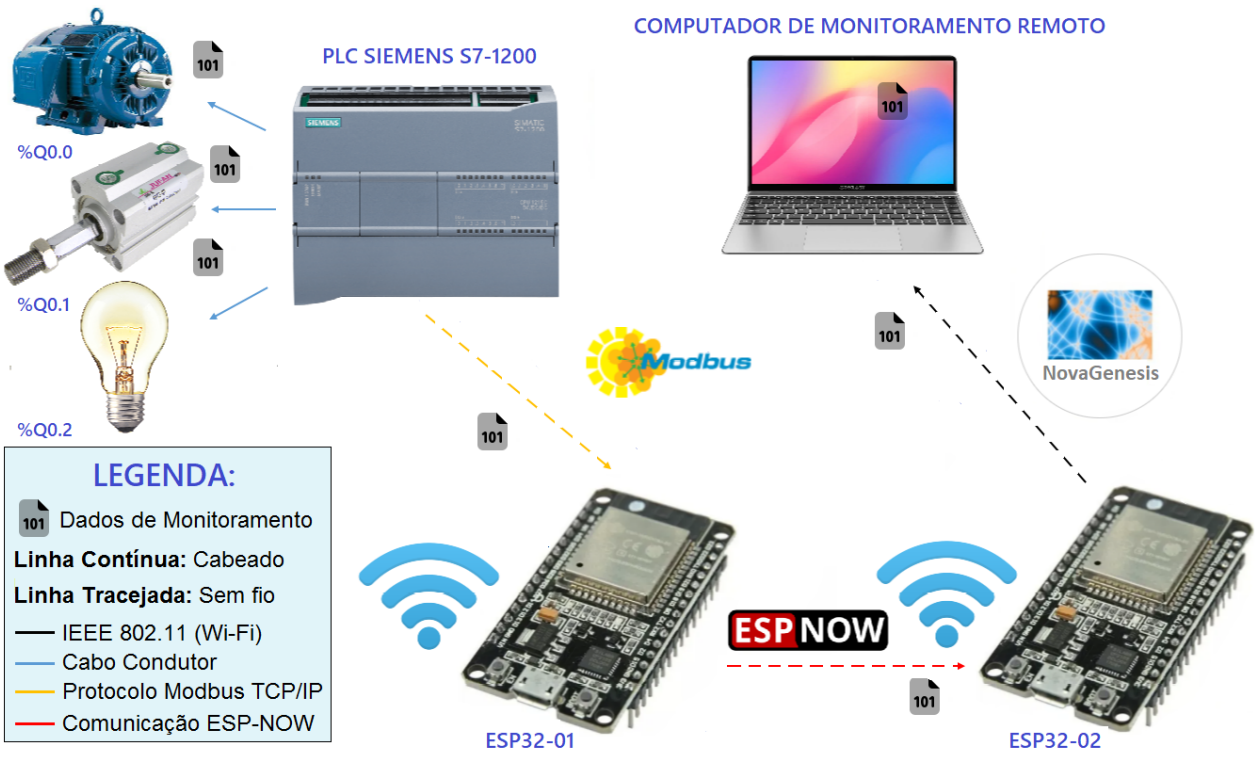

Figura 4.1: Diagrama do Cenário Geral da aplicação NG como CPS em I4.0.

Baseado na Figura 4.1, a Tabela 4.1 lista os equipamentos de *hardware* (Hw) e os *softwares* (Sw) necessários para o cenário da aplicação.

Tabela 4.1: Equipamentos de *hardware* e *software* para a execução do cenário.

No.	Hw	Sw	Equipamento	Quantidade
1	X		PLC Siemens S7-1200 (CPU 1214C DC/DC/DC)	**1**
2	X		Microcontrolador ESP32	**2**
3	X		Computador para monitoramento remoto	**1**
4		X	TIA Portal V13	**1**
5		X	Modbus Poll	**1**
6		X	Arduino IDE 1.8.13	**1**
7		X	Eclipse IDE 2021-03	**1**

Conforme pode ser visto na Figura 4.1, o cenário geral da aplicação cobre: a Parte 1, definida pelo ambiente industrial legado com os equipamentos e protocolo Modbus TCP/IP para comunicação externa, que será usado para criar o ambiente de aplicação para a digitalização de seus ativos; e a Parte 2, sendo a parte inovadora apresentada pela dissertação, com o uso de 2 microcontroladores ESP32 para, respectivamente, a aquisição dos dados fabris legados e

o desenvolvimento do ICPS através da arquitetura FI NG, validando a ideia de que a mesma pode ser desenvolvida baseada no modelo RAMI 4.0 com direcionamento a possíveis soluções relacionadas às limitações da Internet atual. A troca de informações entre os microcontroladores ESP32 é realizada via comunicação ESP-NOW, que é um protocolo proprietário desenvolvido comercialmente pela empresa Espressif. É importante ressaltar que, como esta solução foi proposta utilizando Modbus TCP/IP como protocolo industrial, além de usar o protocolo proprietário de comunicação ESP-NOW, é necessária uma reestruturação de todo o projeto caso o padrão de rede industrial ou os tipos de microcontroladores sejam alterados.

Para o monitoramento remoto, o equipamento controlador utilizado trata-se de um *Programmable Logic Controller* (PLC), em português, Controlador Lógico Programável, Siemens S7-1200, responsável por programar um processo industrial capaz de monitorar uma Tabela de Imagem de Saída (TIS) na memória, também chamado de *buffer*, que reflete e armazena o estado de 3 saídas digitais, e enviar os dados à rede via comunicação Modbus TCP/IP, sendo configurado como Cliente Modbus. Toda a programação necessária é feita em linguagem Ladder através do *software* TIA Portal V13. O Servidor responsável pelo recebimento dos estados das saídas digitais via Modbus é o microcontrolador ESP32-01, programado em linguagem C++ pelo *software* Arduino IDE 1.8.13 devido à sua facilidade de manuseio e bibliotecas internas Modbus e Wi-Fi de fácil interpretação. Além da configuração como Servidor Modbus, o ESP32-01 envia os dados das saídas digitais do PLC via comunicação direta na camada de enlace, denominada ESP-NOW, ao microcontrolador ESP32-02, que por sua vez possui a NG embarcada (EPGS). Por fim, o ESP32-02 transmite os dados coletados ao computador de monitoramento e controle remoto (PGCS) via mensagens NG. Ele é programado em linguagem C++ no *software* Eclipse IDE 2021-03 devido à maior facilidade de uso do mesmo para trabalhar com diferentes arquivos fonte em um mesmo projeto.

O motivo de se usar 2 microcontroladores ESP32 são, primeiramente, permitir o uso conjunto do protocolo Modbus TCP/IP e da NG, visto que enquanto o Modbus TCP/IP trabalha com a camada de transporte, a NG pode trabalhar somente com a camada física/enlace para comunicação, e separar essas tarefas é necessário, já que uma outra solução seria alterar a biblioteca interna Modbus ou NG, cujo grau de complexidade é maior. Além disso, o uso da comunicação ESP-NOW é interessante devido à sua capacidade de comunicar com outros microcontroladores ESP32 usando o endereço MAC do par independente da rede local, realizando a comunicação diretamente através da camada de enlace. Desta forma, em um cenário amplo, uma grande quantidade de PLCs poderiam trocar informações entre si sobre seus dispositivos I/O através de seus respectivos representantes ESP32 acoplados usando comunicação ESP-NOW, permitindo uma maior interação e alternativa de comunicação interna além da NG. A comunicação ESP-NOW será detalhado mais adiante.

O cenário de testes de monitoramento do *buffer* do PLC para que os dados sejam enviados para monitoramento remoto foi realizado no ambiente físico, sendo associados à Registradores Holding do protocolo Modbus, que são armazenados em um *buffer* do programa. O objetivo da criação deste cenário é justamente criar um ambiente industrial real para a realização dos testes de digitalização dos ativos pelo ICPS desenvolvido pela arquitetura NG. Porém, os testes de conexão e comunicação Modbus TCP/IP para transmissão e recepção de dados foi feito

utilizando o *software* Modbus Poll como simulador para o Modbus Cliente, capaz de simular o *buffer* de monitoramento que armazena os estados dos dispositivos I/O do PLC através dos Registradores Holding. O motivo se deve ao fato do PLC Siemens S7-1200 disponível estar localizado no laboratório III-1 do Inatel, e o computador de monitoramento e controle remoto (PGCS) estar disponível no ICT-Lab, tendo cada laboratório uma rede distinta, de difícil interconexão. Desta forma, apesar da possibilidade de utilizar o ambiente físico real com o PLC e o seu respectivo *software* de programação TIA Portal V13, optou-se pelo uso da simulação para poder facilitar o gerenciamento completo da aplicação, controlando todo o cenário a partir de um único equipamento, que nesse caso é um computador com o simulador Modbus Poll instalado.

Para melhor entendimento e organização do Cenário Geral, o mesmo foi dividido em 2 partes. Conforme já detalhado anteriormente, a Parte 1 tem como objetivo mostrar a viabilidade de aquisição de dados dos estados das saídas digitais em um ambiente fabril real através de um *buffer* de monitoramento do PLC, com os ativos industriais físicos e comunicação externa via protocolo Modbus TCP/IP. Já a Parte 2 contém a inovação com a aquisição dos dados do ambiente industrial criado através do *buffer* de monitoramento simulado no *software* Modbus Poll para desenvolvimento do ICPS baseado no modelo RAMI 4.0 através da arquitetura NG. As subseções a seguir detalham as Partes 1 e 2 do projeto.

4.1.1 Cenário Geral - Parte 1: Ambiente Industrial para Teste do ICPS

A Figura 4.2 ilustra o diagrama geral, focando na Parte 1 da aplicação, que é a criação do ambiente industrial para a realização dos testes do ICPS a ser desenvolvido.

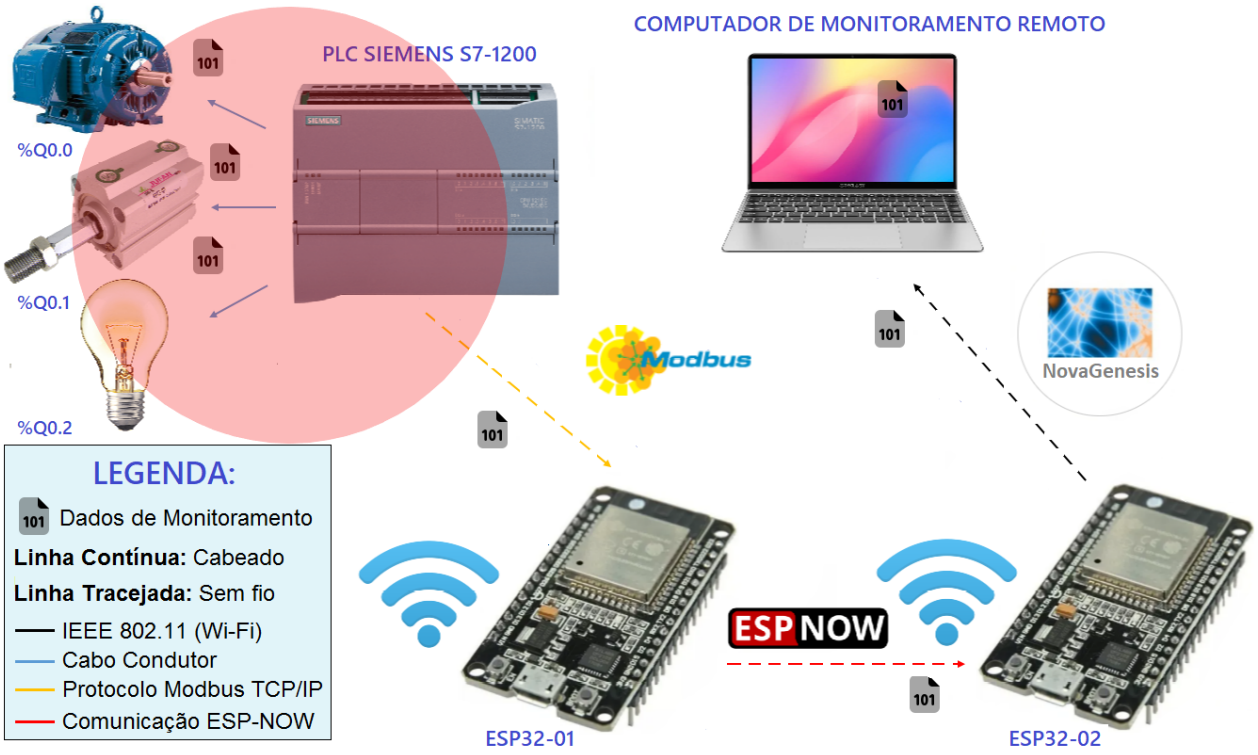

Figura 4.2: Diagrama da aplicação com foco na Parte 1 do Cenário Geral.

Baseado na Figura 4.2, a Parte 1 do Cenário Geral é responsável pelo setor industrial de operação da aplicação, capaz de gerar as ações dos dispositivos de entrada e saída através da programação do processo de manufatura. Além disso, tem como objetivo coletar os dados I/O desejados para monitoramento da rede via protocolo Modbus TCP/IP. Ele é composto pelo equipamento de *hardware* PLC S7-1200 e *software* TIA Portal V13. O PLC S7-1200 tem como objetivo controlar dispositivos de entrada e saída conectados ao seu módulo I/O através do TIA Portal V13, que por sua vez é responsável por criar o programa de execução do processo industrial desejado. A CPU 1214C DC/DC/DC usada no projeto é constituída por 14 entradas digitais, 10 saídas digitais e 1 entrada analógica. Além disso, o PLC também contém o módulo 6ES7232-4HA30-0XB0 referente à saída analógica, porém não será utilizada na experimentação.

Primeiramente, é necessário criar a comunicação Modbus TCP/IP e configurá-la para poder receber e transmitir as informações desejadas pela rede. Para isso, primeiramente foi inserido o bloco de comunicação *MB_Client*. Este bloco é responsável por inicializar e configurar o protocolo Modbus TCP/IP como cliente (mestre) e comunicar com possíveis servidores (escravos) conectados à rede. É válido ressaltar que tanto o cliente, quanto o servidor podem enviar ou receber dados de seu respectivo par, pois a definição tem como objetivo apenas definir o dispositivo responsável por iniciar a comunicação e enviar as requisições desejadas

(cliente) e responder às requisições (servidor). Como o PLC Siemens S7-1200 foi definido como cliente, ele enviará as requisições ao escravo com os dados coletados das saídas digitais e solicitações dos comandos de controle. O escravo, por sua vez, responderá às requisições sinalizando se a entrega foi efetuada com sucesso. A Figura 4.3 ilustra o bloco *MB_Client*.

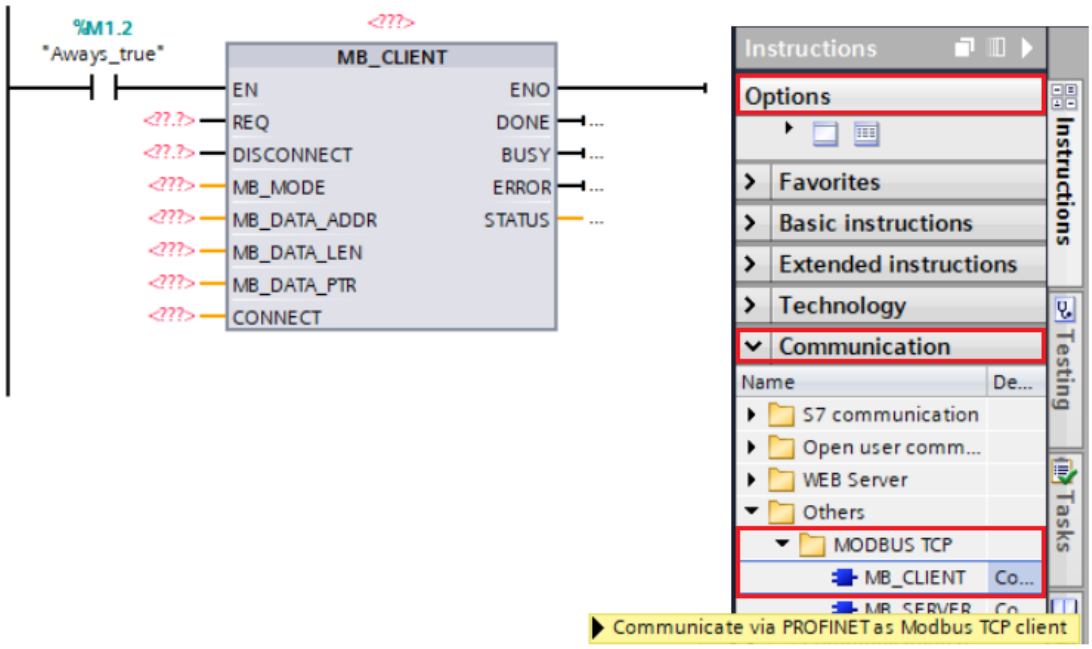

Figura 4.3: Bloco *MB_Client* para inicialização e configuração Modbus TCP/IP.

Conforme mostrado na Figura 4.3, uma vez inicializado o bloco *MB_Client*, é necessário configurá-lo para sua comunicação com o servidor par. Similar ao *MOVE*, este bloco também possui um pino *EN* que deve receber nível lógico alto para seu correto funcionamento. Para isso, foi inserido um contato normalmente aberto atrelado à %M1.2, de modo que o valor "1" habilita o pino. No TIA Portal, assim como a inicial I indica uma entrada e a inicial Q indica uma saída, a inicial M refere-se a um bit de memória. Esse bit de memória pode ser pré-definido e qualquer valor pode ser alocado, e desta forma, sempre que o mesmo for referenciado no código, seu valor é automaticamente fornecido de acordo com a configuração pré-estabelecida. Sabendo disso, o bit de memória %M1.2 foi pré-definido como "1", e portanto o pino *EN* sempre estará habilitado, a menos que o bit de memória indicado seja alterado.

Para a configuração dos demais pinos do bloco *MB_Client*, foi criado um bloco de dados denominado *MB_Config* e posteriormente cada endereço estático do bloco foi direcionado a seu respectivo pino do Cliente Modbus TCP para passar a devida informação. A Figura 4.4 ilustra o bloco de dados *MB_Config* e seus endereços estáticos alocados ao bloco *MB_Client*.

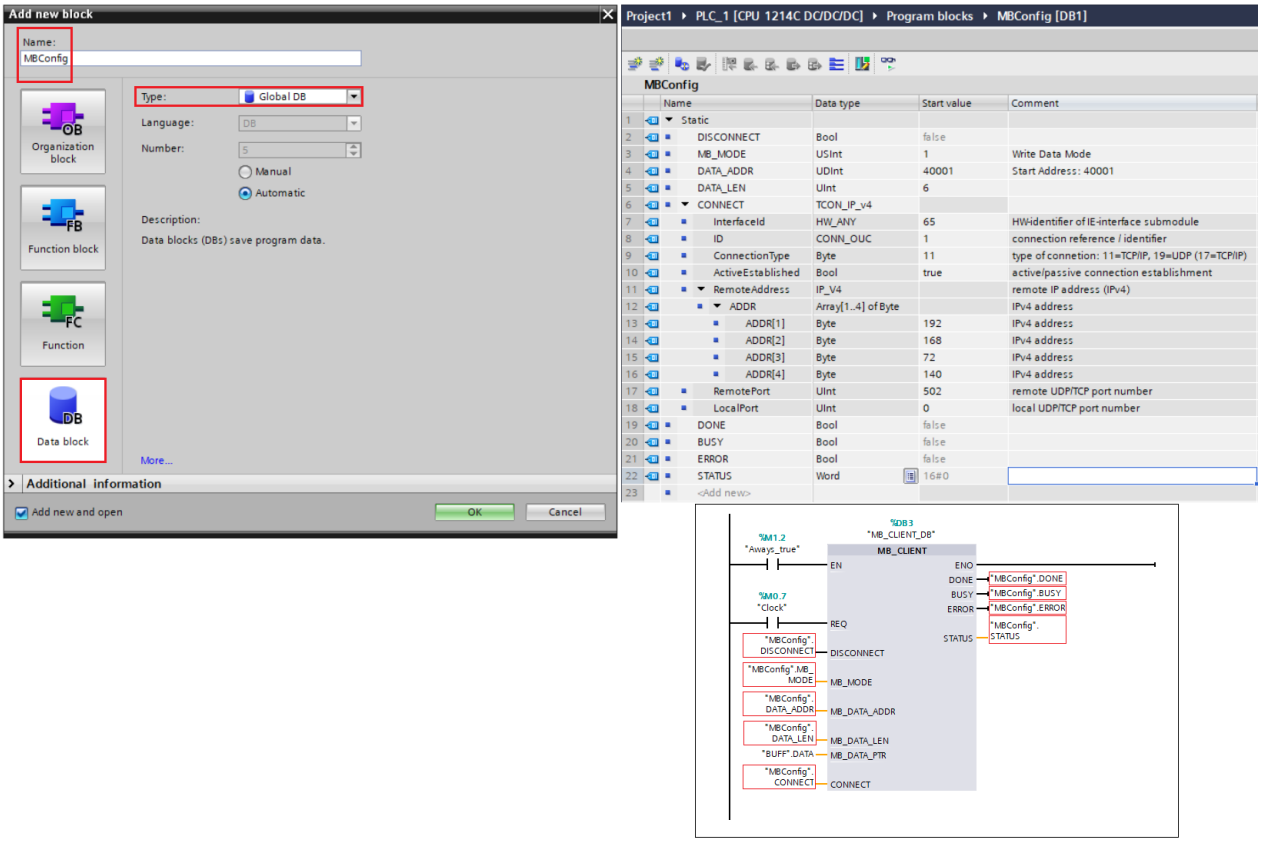

Figura 4.4: Configuração do Bloco *MB_Config* e direcionamento dos dados aos respectivos pinos do Bloco Modbus_Client.

Conforme ilustrado na Figura 4.4, o primeiro valor do bloco de dados consiste na variável *DISCONNECT* do tipo *booleana* (Bool), configurada inicialmente como "false" para permitir a inicialização Modbus. A variável *MB_MODE* do tipo inteiro não sinalizado de 8 bits (USInt) determina o modo de operação do dispositivo, definido como "1", referente ao modo de escrita de dados. O campo *DATA_ADDR* do tipo inteiro não sinalizado de 32 bits (UDInt) configura o endereço inicial Modbus. Baseado nas informações descritas na Tabela 2.1 presente na Seção Modbus TCP/IP do Capítulo 2, o valor "40001" refere-se ao prefixo "4", cujo bloco de memória corresponde aos Registradores Holding, seguidos pelo endereço "0001". Os dados foram escolhidos para serem transmitidos em Registradores Holding devido à sua característica de leitura/escrita para saídas digitais, conforme visto na Tabela 2.1. A variável *DATA_LEN* do tipo inteiro não sinalizado (UInt) atribui o comprimento dos dados a serem enviados, sendo o número de Registradores Holding usados. Para esta aplicação, será realizado o monitoramento do *buffer* referente ao estado de 3 saídas digitais do PLC para monitoramento. Desta forma, os 3 primeiros Registradores Holding são usados para armazenar as informações das saídas %Q0.0, %Q0.1 e %Q0.2 e serem enviadas pela rede

para monitoramento remoto.

O campo *CONNECT* possui a peculiaridade de atribuição do tipo TCON_IP_v4, e é responsável por configurar a rede em si. Primeiramente, a variável InterfaceId do tipo HW_ANY define o identificador de *hardware*, que nesse caso é "65". Para verificar o identificador de *hardware*, basta acessar o módulo da interface PROFINET do PLC, conforme ilustrado na Figura 4.5.

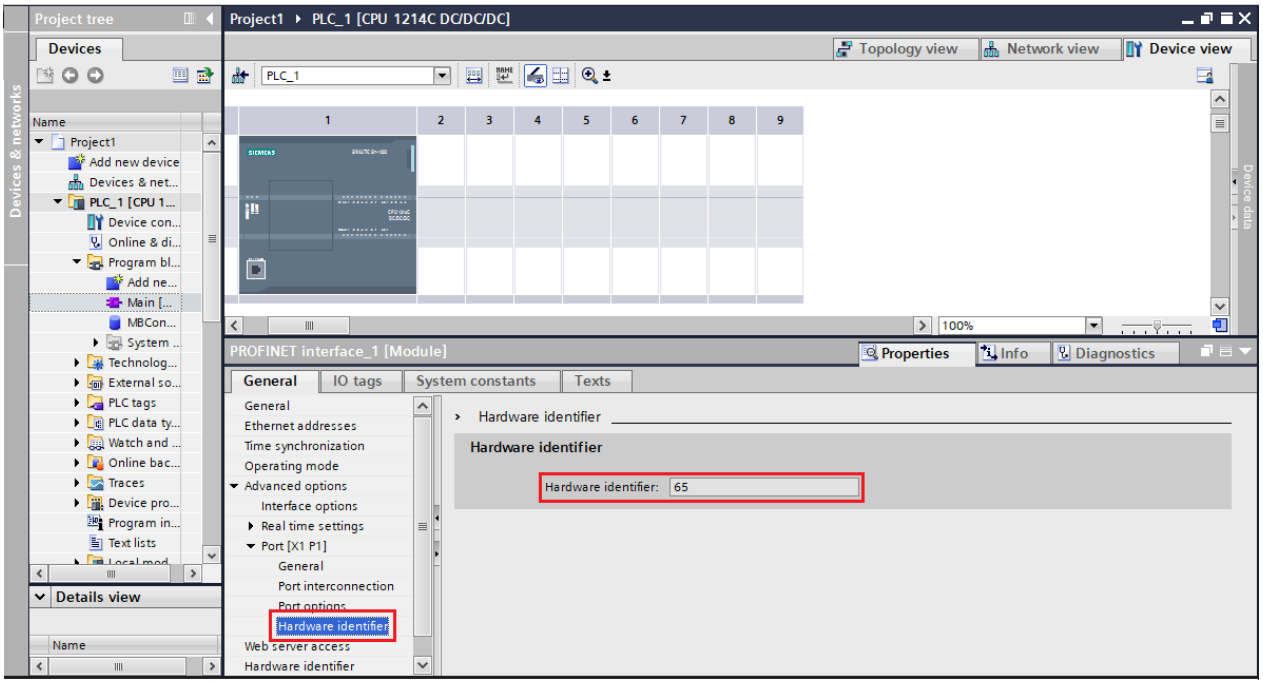

Figura 4.5: Identificador de *Hardware* do PLC.

Ainda no campo *CONNECT*, foi configurada a variável *ID* do tipo CONN_OUC de valor "1", que corresponde ao identificador da conexão. a variável *ConnectionType* do tipo Byte determina o tipo de conexão, como por exemplo "11" para TCP/IP e "19" para UDP. Para esta aplicação, foi utilizado o parâmetro "11". A variável *ActiveEstablished* do tipo *booleana* foi definida como "true" para permitir o estabelecimento de conexão ativa. Já a variável *RemoteAddress* do tipo IP_V4 é destinada a alocar o endereço IP remoto, que por sua vez pertence ao IP do ESP32-01 da aplicação, que atuará como Modbus Server. As variáveis *LocalPort* e *RemotePort* do tipo UInt definem, respectivamente, o número da porta TCP local, configurada como "0", e o número da porta TCP remota, que por padrão é configurada como "502" para aplicações Modbus.

Por fim, os campos *booleanos DONE*, *BUSY*, *ERROR* foram configurados inicialmente como "false" e o campo *STATUS* do tipo Word foi definido para inicializar em "0". Esses campos podem retornar um valor diferente de acordo com a execução do processo, como por exemplo um Word referente ao status atual ou o valor de *ERROR* para "true" no caso de

um erro de comunicação.

Com todos os campos preenchidos, foi necessária a alocação de cada um deles em seu respectivo pino do bloco *MB_Client*. Para isso, basta utilizar a sintaxe (”nome do bloco de dados”.campo do bloco). Como exemplo, no pino *DISCONNECT*, foi inserida a sintaxe ”MBConfig”.DISCONNECT, e o mesmo foi aplicado aos demais pinos.

Ao observar a Figura 4.3, pode-se notar que não foram detalhados os pinos *REQ* e *MB_DATA_PTR*. O pino *REQ* determina o tempo em que as requisições serão enviadas pelo cliente ao servidor. Nele é definido um bit de memória (%M0.7) para um relógio que determinará o ciclo de requisições. Este clock pode ser definido de acordo com a aplicação desejada do usuário. Já o pino *MB_DATA_PTR* deve ser direcionado a um buffer que irá conter os dados dos Registradores Holding. Para isso, foi criado um bloco de dados usado como *buffer* de armazenamento e inserido no pino *MB_DATA_PTR* do bloco *MB_Client*, conforme ilustrado na Figura 4.6.

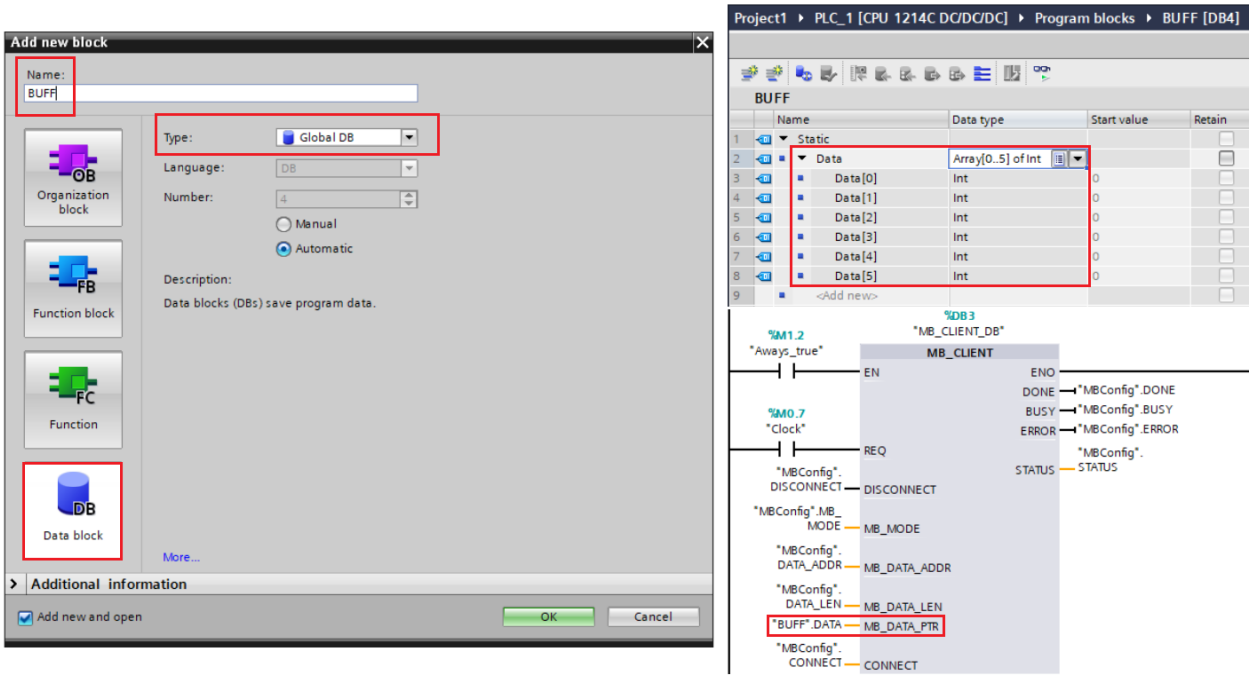

Figura 4.6: Bloco de Dados do *Buffer* para armazenamento dos dados das saídas digitais do PLC para envio e dos comandos de controle recebidos.

Este bloco possui o nome *”BUFF”* e comprimento igual a 6, sendo uma faixa de valores inteiros de 0 a 5. Caso seja necessário o monitoramento de uma saída que entregue um valor diferente do tipo ”inteiro”, basta alterar o tipo de dado definido como *int* presente na coluna *Data Type*, ilustrada na Figura 4.6, para o tipo necessário de acordo com a aplicação. Cada posição dessa faixa de valores refere-se a um Registrador Holding, cujos nomes são, respectivamente: *”BUFF”.DATA[0]*, *”BUFF”.DATA[1]*, *”BUFF”.DATA[2]*,

"BUFF".DATA[3], *"BUFF".DATA[4]* e *"BUFF".DATA[5]*. Para esta aplicação, as 3 primeiras posições do *buffer* foram vinculadas à saídas digitais do PLC %Q0.0, %Q0.1 e %Q0.2 para monitoramento remoto. Assim sendo, quando a comunicação Modbus TCP/IP é inicializada, os status das saídas %Q0.0, %Q0.1 e %Q0.2 são inseridos nas posições *"BUFF".DATA[0]*, *"BUFF".DATA[1]* e *"BUFF".DATA[2]* do *buffer*, e enviados para o ESP32-01 armazenar os dados de leitura.

Com toda a comunicação Modbus TCP/IP configurada, é necessário associar as posições do *buffer* de dados *"BUFF"* criado anteriormente com cada uma das saídas digitais do PLC. Para o cenário da aplicação, foi desenvolvido um processo industrial para monitoramento do *buffer* que reflete no estado das saídas digitais. Então. foi criada uma lógica na qual o acionamento de uma entrada implica no acionamento de uma saída, considerando 3 entradas digitais (Chaves 0, 1 e 2) responsáveis por acionar, respectivamente, 3 saídas digitais (ex.: motor, pistão e lâmpada). Então, os status (ON/OFF) dessas saídas são inseridos em *"BUFF".DATA[0]*, *"BUFF".DATA[1]* e *"BUFF".DATA[2]*, respectivamente, e enviados pela rede via protocolo Modbus TCP/IP para leitura.

A Figura 4.7 ilustra a parte do programa responsável pelo monitoramento do *buffer* que armazena os estados das saídas digitais %Q0.0, %Q0.1 e %Q0.2. Como pode ser visto, seus estados são manipulados para serem armazenados nas posições *"BUFF".DATA[0]*, *"BUFF".DATA[1]* e *"BUFF".DATA[2]* do *buffer*. O programa foi feito em linguagem Ladder através do *software* TIA PORTAL V13.

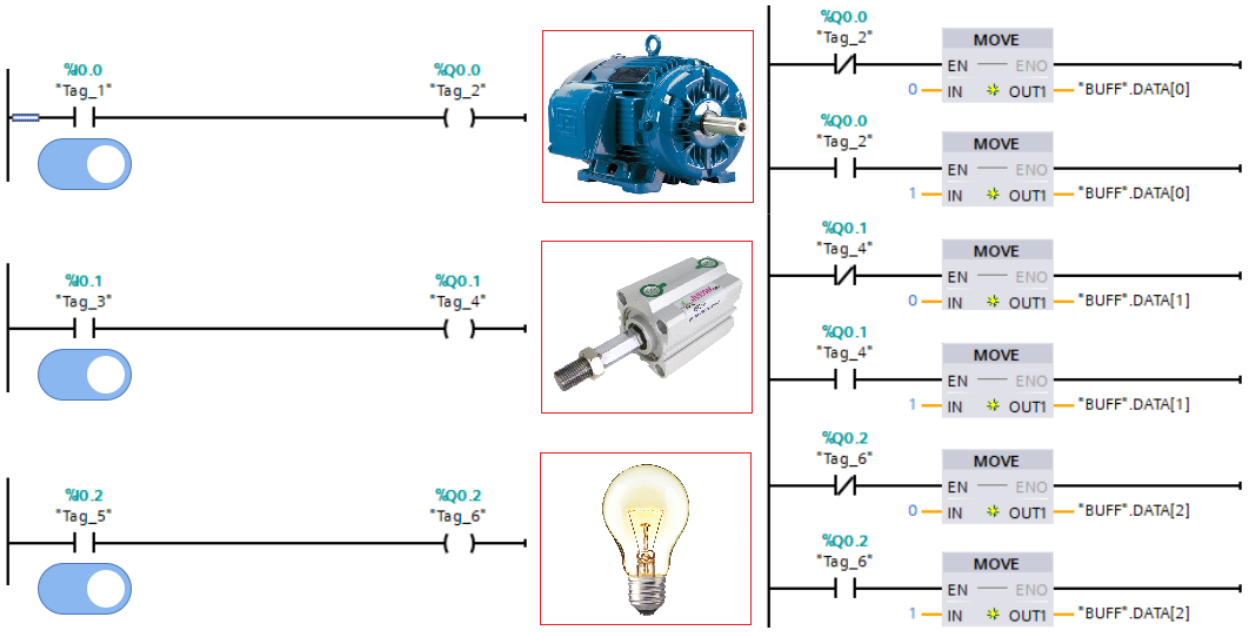

Figura 4.7: Programa Ladder do processo industrial para monitoramento remoto.

Como pode ser visto na Figura 4.7, quando a entrada digital %I0.0 (Chave 0) está em

nível lógico alto (1), a saída digital %Q0.0 (Motor) é acionada; caso contrário, a saída é desligada. O mesmo procedimento vale para %I0.1 (Chave 1) e %Q0.1 (Pistão); e %I0.2 (Chave 2) e %Q0.2 (Lâmpada).

Para poder enviar os dados dos estados (ON/OFF) das 3 saídas digitais (%Q0.0, %Q0.1 e %Q0.2) para a rede via protocolo Modbus TCP/IP, é necessário inserí-los em um bloco de dados como *buffer*. Porém, por se tratarem de saídas digitais, seus valores são *booleanos* e não podem ser movidos dentro do programa. Para isto, foi necessária a implementação do bloco *MOVE*, que tem o objetivo de mover um determinado valor (ex.: valor inteiro) inserido no pino de entrada *IN* para uma variável no pino de saída *OUT1*. O bloco *MOVE* transmite o valor desejado a uma variável apenas quando seu pino *ENABLE* está habilitado. Com isto, como pode ser observado na Figura 4.7, para cada contato normalmente fechado no pino *ENABLE*, o nível lógico alto ocorre quando as saídas estão desacionadas (valor *booleano* 0), e consequentemente o valor inserido no pino *IN* (valor inteiro 0) é enviado para o pino *OUT1*, que possui o *buffer* (*"BUFF".DATA*) seguido da respectiva posição de saída (*"BUFF".DATA[0]* para %Q0.0, *"BUFF".DATA[1]* para %Q0.1 e *"BUFF".DATA[2]* para %Q0.2). Da mesma forma, para cada contato normalmente aberto em *ENABLE*, é esperado nível lógico alto quando as saídas estão acionadas, e então o valor inserido no pino *IN* (valor inteiro 1) é movido para o *buffer* em *OUT1*. Essa manipulação de dados é capaz de "transformar"os valores *booleanos* "0" e "1" das saídas digitais em valores inteiros "0"e "1"para serem inseridos nas posições do *buffer*, e finalmente permitir seu envio de status para o monitoramento remoto.

Terminada todas essas etapas, o cliente PLC finalmente está preparado para se comunicar com o servidor através do protocolo Modbus TCP/IP, que para esta aplicação é definido como o microcontrolador ESP32-01. Conforme descrito no inicio do Capítulo, a fim de criar todo o cenário em um único ambiente, foi usado o *software* Modbus Poll para simular o PLC SIEMENS S7-1200 e o *buffer* de armazenamento dos estados das saídas digitais como Cliente Modbus através dos correspondentes Registradores Holding e realizar os testes esperados. A seguir, será descrita a Parte 2 do Cenário Geral, que consiste na aquisição dos dados do ambiente fabril real descrito nesta subseção para o desenvolvimento de um ICPS baseado no modelo RAMI 4.0 através da arquitetura FI NG.

4.1.2 Cenário Geral - Parte 2: Aquisição de Dados do Ambiente Industrial Real e Desenvolvimento do ICPS

Conforme já discutido nas subseções anteriores, a Parte 2 do Cenário Geral apresenta as novidades desta dissertação, cujo foco principal é o desenvolvimento de um ICPS baseado no modelo RAMI 4.0 através da arquitetura FI NG. Porém, para desenvolver este sistema, primeiramente é necessário realizar a aquisição dos dados dos ativos disponíveis no ambiente industrial real criado na Parte 1. Portanto, para facilitar o entendimento do leitor, a Parte 2 do Cenário Geral será dividida em: Parte 2.1, responsável pela aquisição dos dados do ambiente fabril real; e Parte 2.2, destinada ao desenvolvimento do ICPS com o uso da NG.

Cenário Geral: Parte 2.1 - Aquisição de Dados do Ambiente Industrial Real

A Figura 4.8 mostra o diagrama geral com foco na Parte 2.1 do projeto:

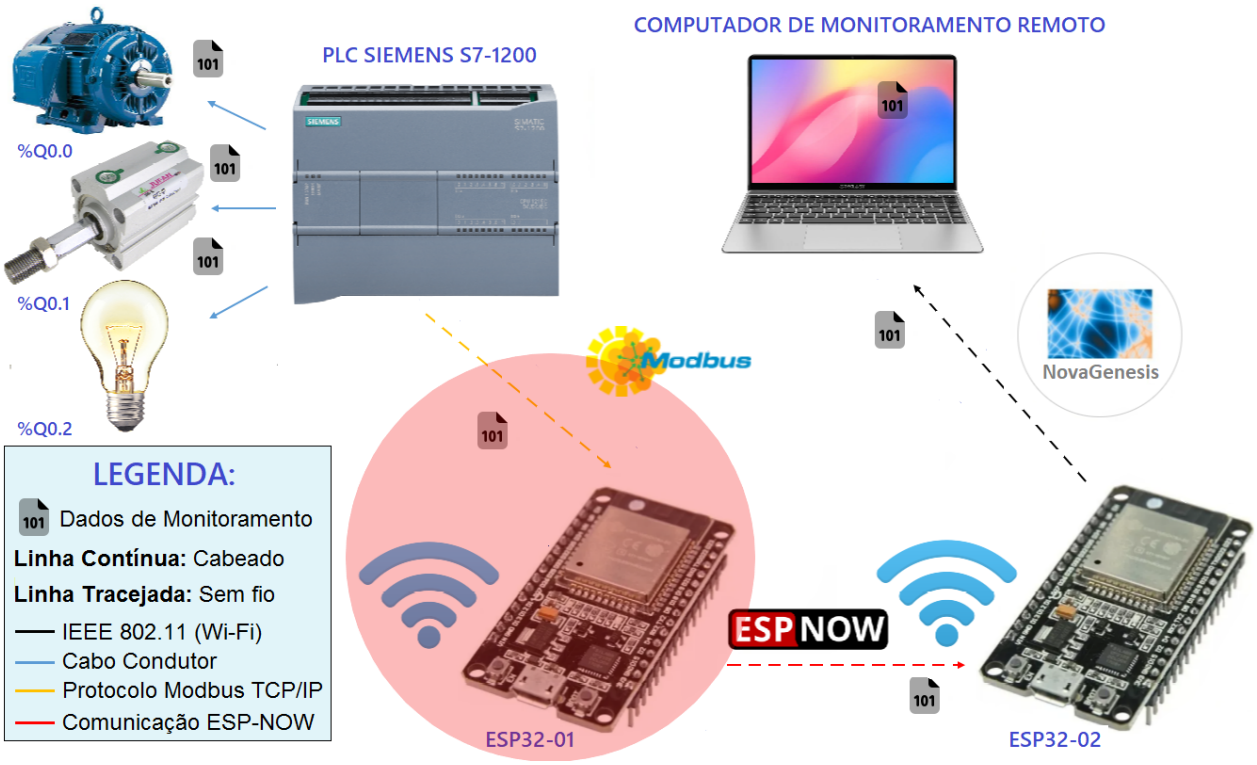

Figura 4.8: Diagrama da aplicação com foco na Parte 2.1 do Cenário Geral.

Conforme já detalhado anteriormente, a Parte 2.1 do Cenário Geral tem como principal objetivo realizar a coleta de dados do ambiente fabril (Parte 1 do Cenário Geral) para que eles possam ser utilizados poseteriormente no modelo ICPS a ser desenvolvido pela NG. Assim sendo, o *software* Arduino IDE 1.8.13 foi usado para programar em linguagem C ++ o microcontrolador ESP32-01, configurado como servidor Modbus e transmissor/receptor ESP-NOW para a função de monitoramento. Esta função, por sua vez, é responsável por receber o *buffer* com os estados das saídas digitais %Q0.0, %Q0.1 e %Q0.2 do cliente PLC via Modbus TCP/IP, realizar o tratamento e enviá-los ao ESP32-02 via comunicação ESP-NOW. A programação foi realizada para enviar estes estados periodicamente, sendo os valores alterados de forma manual. Esta etapa deve ser feita para poder realizar a entrega dos dados à NG diretamente pela usando o Wi-Fi através da comunicação entre os microcontroladores via comunicação ESP-NOW, sem a necessidade de uso da pilha TCP/IP para tal.

A Figura 4.9 ilustra o fluxograma da programação do ESP32-01, cujo código completo desenvolvido está disponível no Apêndice A.1.

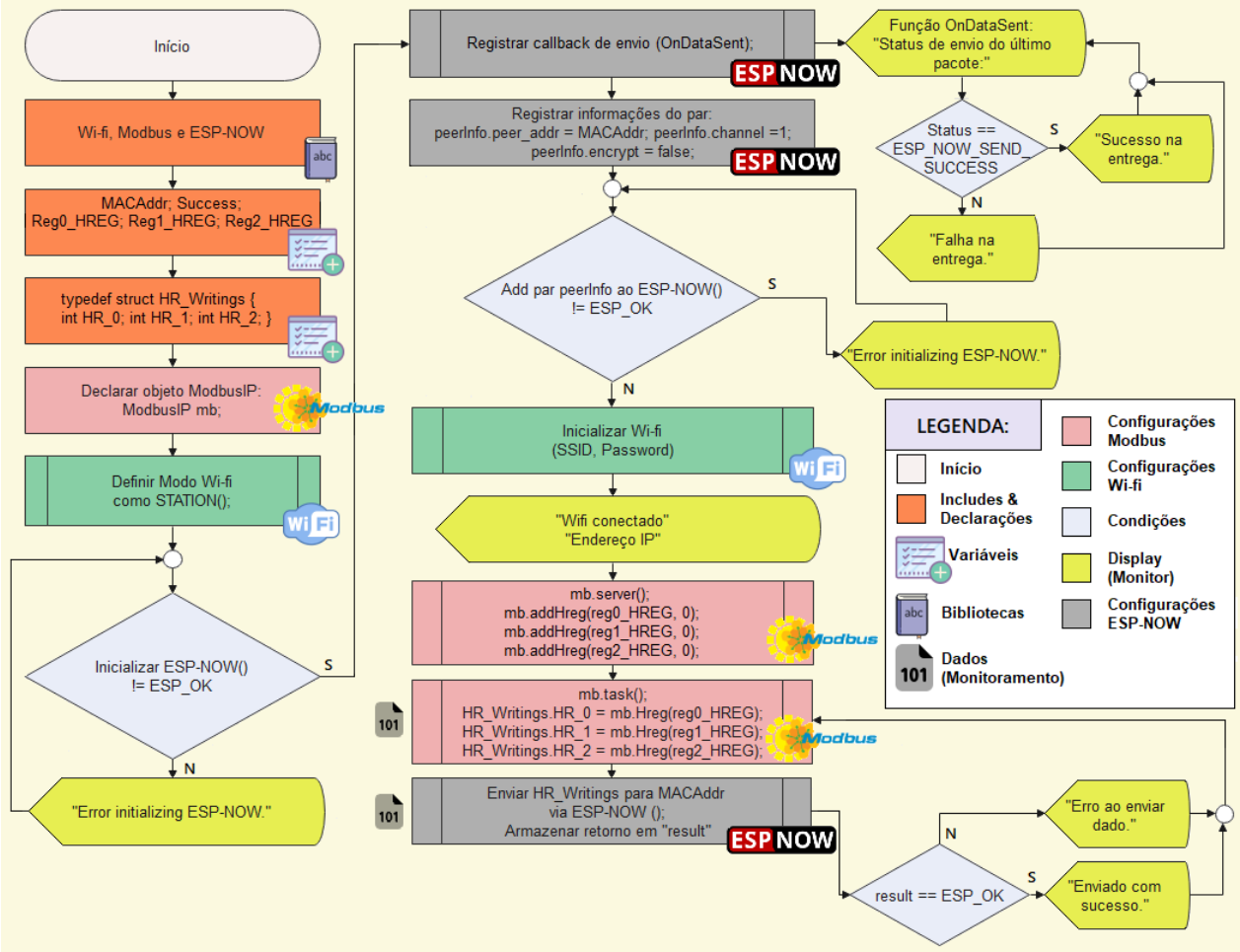

Figura 4.9: Fluxograma do código do ESP32-01.

Primeiramente, as bibliotecas necessárias para o funcionamento do código são incluídas. Essas bibliotecas correspondem à Wi-Fi, Modbus e ESP-NOW e permitem o uso de suas respectivas funções e tarefas. Em relação à declaração de variáveis e objetos, são adicionadas as seguintes componentes:

1. **MACAddr:** variável para armazenar o endereço MAC do receptor ESP-NOW. Para este projeto, corresponde ao endereço MAC do ESP32-02;

2. **Success:** variável usada para armazenar a situação atual de entrega de dados ESP-NOW, como sucesso ou falha;

3. **reg0_HREG, reg1_HREG e reg2_HREG:** variáveis de offset para referenciar os 3 Registradores Holding para armazenar a leitura das saídas digitais;

4. **HR_Writings:** variável do tipo Struct com 3 valores inteiros disponíveis para armazenamento interno (HR_0, HR_1 e HR_2), usada para enviar os dados recebidos do PLC via comunicação Modbus TCP/IP para o ESP32-02 via comunicação ESP-NOW;

5. **ModbusIP mb:** objeto ModbusIP de nome mb, usado como referência para ser aplicado nas funções Modbus.

Na seção de *Setup* da IDE do Arduino, o primeiro passo é definir o modo Wi-Fi como *Station* para o correto funcionamento da comunicação ESP-NOW, que é inicializado em seguida. Com a comunicação ESP-NOW em operação, é registrada a *callback* de envio denominada como *OnDataSent*. A tarefa *OnDataSent* é usada para monitorar o status de envio do último pacote retornando sucesso ou falha na entrega, e é executada em loop infinito paralelo ao restante do código. Além disso, é necessário destacar que a *callback* de envio da comunicação ESP-NOW é executada a partir de uma tarefa Wi-Fi de alta prioridade, e portanto é recomendado que não sejam feitas operações demoradas na função de retorno de chamada. Em vez disso, o ideal é publicar os dados necessários em uma fila e tratá-los a partir de uma tarefa de prioridade mais baixa.

Para realizar o pareamento entre os microcontroladores ESP32, é necessário registrar as informações do par que receberá os dados do ESP32-01, que por sua vez corresponde ao ESP32-02 desta aplicação. Essas informações referem-se a:

1. **peerInfo.peer_addr:** refere-se ao endereço MAC do par, recebendo nesse caso a variável MACAddr descrita anteriormente. O endereço MAC é usado como identificador para poder reconhecer o dispositivo par com o qual o microcontrolador irá se comunicar, que neste caso refere-se ao ESP32-02;

2. **peerInfo.channel:** refere-se ao canal utilizado para a comunicação ESP-NOW entre os dispositivos ESP32. É importante notar que, no caso do uso simultâneo da comunicação ESP-NOW e da comunicação Wi-Fi, o canal do roteador usado para a rede sem fio deve ser o mesmo configurado nesta variável. Para este projeto, o canal escolhido foi "1", e portanto o canal do roteador para a rede sem fio deve ser configurado de forma similar. A Figura 4.10 ilustra a configuração do roteador Wi-Fi para uso do Canal 1.

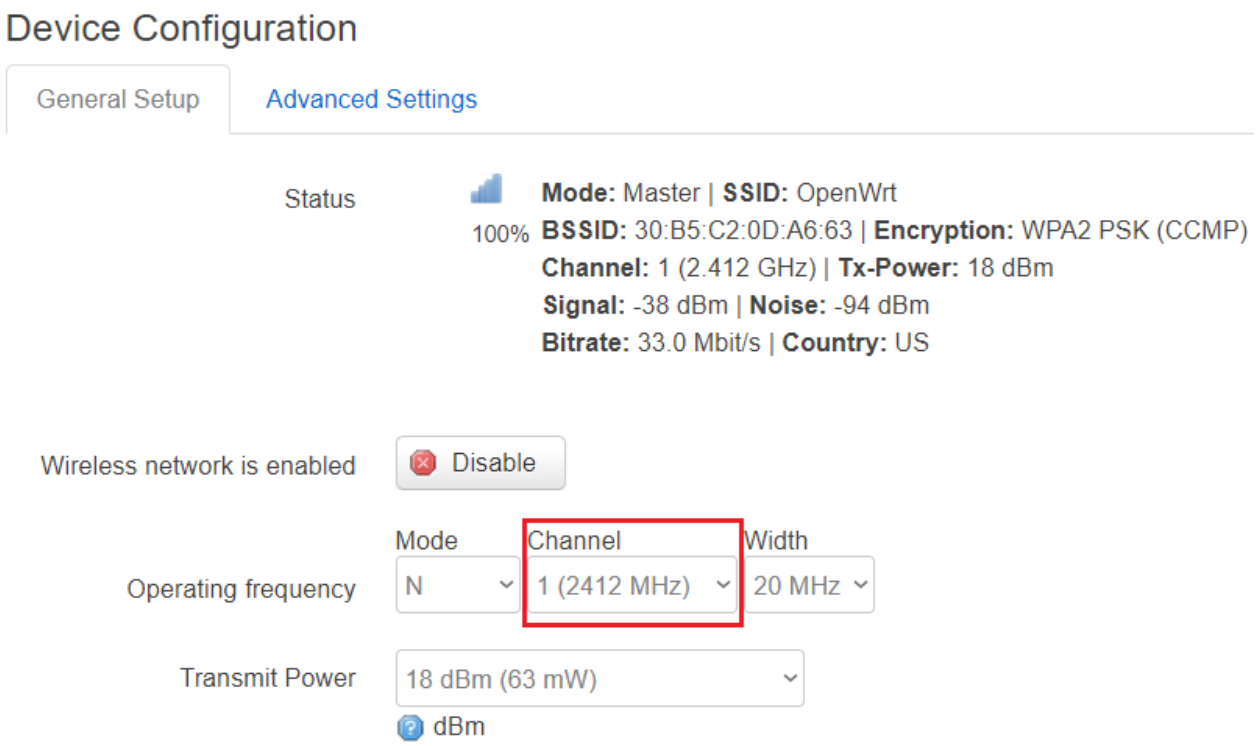

Figura 4.10: Configuração do Roteador Wi-Fi para o Canal 1.

3. **peerInfo.encrypt:** refere-se ao uso de encriptação dos dados. Neste caso, como os dados não serão encriptados, a variável recebe o valor "false".

Dadas as informações necessárias, o par é adicionado para poder se comunicar via ESP-NOW, e então a função de inicialização do Wi-Fi pode ser chamada, definindo o *SSID* e *Password* da rede. Visto que a conexão à uma rede Wi-Fi não é necessária para a comunicação dos microcontroladores no comunicação ESP-NOW, a mesma é utilizada neste caso para poder receber os dados via Modbus TCP/IP. Então, o objeto Modbus denominado *mb* declarado no início do programa é usado como referência para as chamadas das funções Modbus, que são:

1. **mb.server():** chamada para configurar o dispositivo como servidor Modbus, de modo a se comunicar com seu respectivo cliente, que no caso desta aplicação é o PLC;

2. **mb.addHreg(reg0_HREG, 0):** chamada para adicionar um Registrador Holding, referenciando seu offset à variável *reg0_HREG* e inicializando com valor "0";

3. **mb.addHreg(reg1_HREG, 0):** chamada para adicionar um Registrador Holding, referenciando seu offset à variável *reg1_HREG* e inicializando com valor "0";

4. **mb.addHreg(reg2_HREG, 0):** chamada para adicionar um Registrador Holding, referenciando seu offset à variável *reg2_HREG* e inicializando com valor "0";

Finalizada a seção *Setup*, a seção *Loop* da IDE do Arduino pode ser configurada, cujas funções são:

1. **mb.task():** chamada da tarefa responsável por inicializar a comunicação Modbus e receber os dados;
2. **HR_Writings.HR_0 = mb.Hreg(reg0_HREG):** armazena o valor do Registrador Holding com offset em *reg0_HREG* (referente à saída %Q0.0 do PLC) na variável do tipo inteiro HR_0 da struct *HR_Writings*;
3. **HR_Writings.HR_1 = mb.Hreg(reg1_HREG):** armazena o valor do Registrador Holding com offset em *reg1_HREG* (referente à saída %Q0.1 do PLC) na variável do tipo inteiro HR_1 da struct *HR_Writings*;
4. **HR_Writings.HR_2 = mb.Hreg(reg2_HREG):** armazena o valor do Registrador Holding com offset em *reg2_HREG* (referente à saída %Q0.2 do PLC) na variável do tipo inteiro HR_2 da struct *HR_Writings*;

Finalmente, após a comunicação Modbus e armazenamento de dados serem estabelecidas corretamente, a função de envio de dados da comunicação ESP-NOW pode ser chamada. Essa função envia a struct *HR_Writings* com os dados das saídas digitais do PLC ao par ESP32-02 que corresponde ao endereço MAC registrado. A fim de monitorar esse envio, este retorno é armazenado em uma variável denominada *result*, que verifica se o dado foi enviado com sucesso ou se houve erro na transmissão.

Para a Parte 2.2 do Cenário Geral, é necessário configurar o ESP32-02 como receptor da comunicação ESP-NOW para receber os dados do ESP32-01, realizar o tratamento e encapsulá-los em mensagens NG para finalmente enviá-los ao computador de monitoramento e controle remoto.

Devido ao fato do PLC Siemens S7-1200 ser disponibilizado pelo Inatel no Laboratório III-1, enquanto que o computador de monitoramento remoto com o PGCS da NG está localizado no Laboratório ICT-Lab, o *software* Modbus Poll foi utilizado na aplicação. Este *software* tem como objetivo simular o cliente Modbus TCP/IP (PLC) e os Registradores Holding (vinculados às saídas digitais). O objetivo da simulação é permitir que o cenário geral seja testado em um único ambiente e com o menor número de recursos possíveis para facilitar a organização e gerenciamento dos testes e resultados, além de englobar o projeto em uma única rede, visto que cada laboratório está configurado com uma rede distinta. Assim, é necessário apenas um único equipamento para executar a Parte 1 do Cenário Geral, sendo neste caso um computador com o simulador Modbus Poll. Desta forma, através da programação do ESP32-01, o cenário de tráfego foi definido para receber um pacote de dados referente ao *buffer* monitorando os 3 estados digitais periodicamente a cada 6 segundos, sendo a alteração de seus valores para teste feita manualmente através do próprio Modbus Poll.

A Parte 2.2 do Cenário Geral será descrita na próxima subseção.

Cenário Geral: Parte 2.2 - Uso da Arquitetura FI NG para o Desenvolvimento do ICPS baseado no RAMI 4.0:

A Figura 4.11 representa o diagrama geral, destacando a Parte 2.2 do Cenário Geral da proposta:

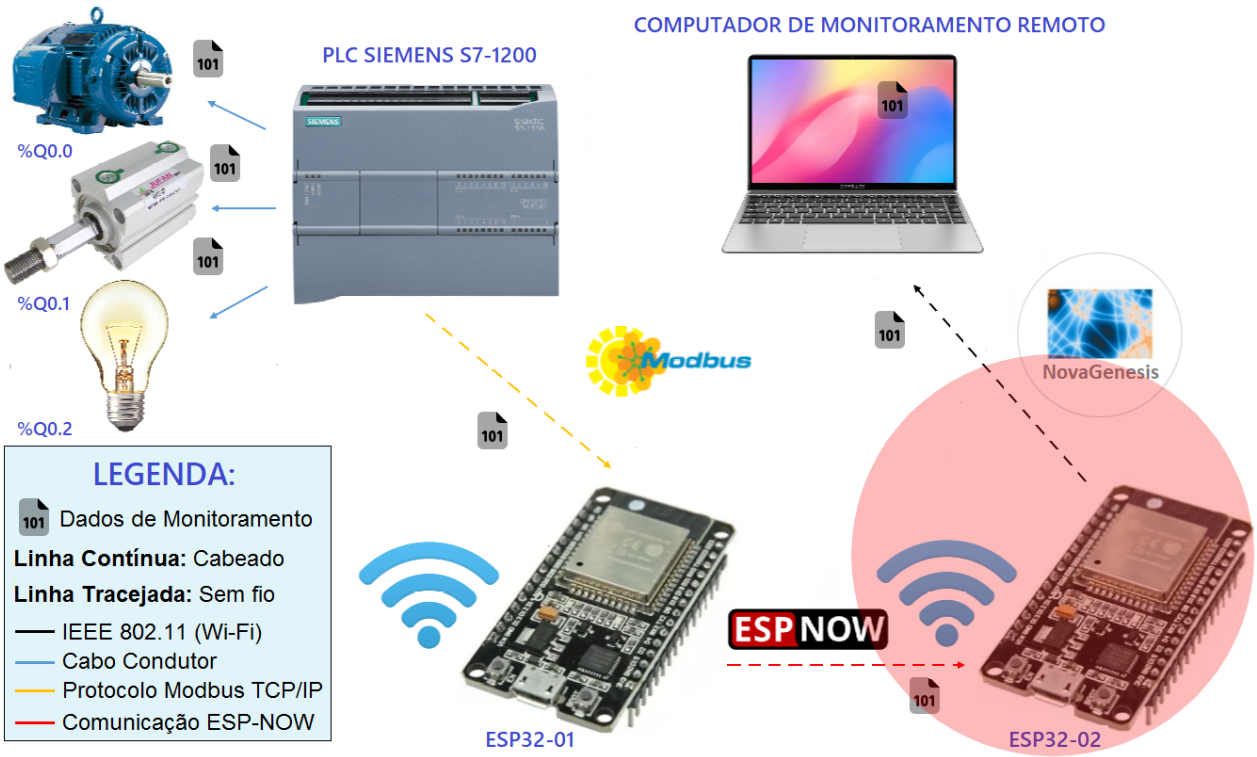

Figura 4.11: Diagrama da aplicação com foco na Parte 2.2 do Cenário Geral.

Para a Parte 2.2 do Cenário Geral, o *software* Eclipse IDE 2021-03 foi utilizado para programar através da linguagem C++ o microcontrolador ESP32-02, configurado como receptor ESP-NOW para receber os dados de leitura do ESP32-01. O motivo da alteração do *software* utilizado para programação no microcontrolador foi o fato da NG possuir uma estrutura de código com diferentes arquivos, e para usar o Arduino IDE 1.8.13 seria necessário adaptar todo o código em um único arquivo, tornando a implementação mais complexa, diferentemente do *software* Eclipse IDE 2021-03, onde podem haver mais de um arquivo trabalhando em conjunto em um mesmo projeto. Além disso, o chip foi embarcado com o EPGS da NG, cujo código permite o funcionamento da arquitetura no dispositivo, e alterado para executar a aplicação da proposta. O objetivo desta aplicação é desenvolver um ICPS embarcado com a arquitetura de Internet do Futuro NG para, além de atender aos requisitos do modelo de

referência RAMI 4.0, lidar com problemas relacionados às limitações da Internet atual, com o diferencial de sua característica de estrutura de nomeação capaz de aproximar o homem à máquina através da sua capacidade de permitir ao usuário trabalhar com linguagem natural, o que direciona a arquitetura aos conceitos primários da Indústria 5.0. O motivo de se embarcar o programa em um microcontrolador se deve ao fato da necessidade da aplicação ser executada em um dispositivo de pequeno porte e discreto, que possa ser alocado no ambiente industrial próximo ao equipamento que se deseja digitalizar, visto que o mesmo será seu representante. O modelo ESP32 foi escolhido para atender a todos os requisitos necessários deste projeto, como por exemplo a capacidade de memória para alocar o programa, o módulo Wi-Fi para poder realizar a comunicação com o computador de monitoramento remoto, e principalmente seu diferencial de possuir um protocolo único denominado ESP-NOW capaz de comunicar diretamente com o microcontrolador responsável pela aquisição de dados dos ativos do ambiente industrial (ESP32-01) sem a necessidade da camada de transporte, já que a NG não faz uso da mesma. Desta forma, é possível se comunicar com o computador de monitoramento remoto enviando mensagens NG com os dados das saídas digitais para monitoramento. A Figura 4.12 mostra o Diagrama de programação do ESP32-02, cujo código completo está disponível no Apêndice B. Para facilitar a compreensão do programa, suas principais funcionalidades foram representadas no diagrama por ícones específicos, cujo significado pode ser visto no mapeamento da Figura 4.13. Este mapeamento pode ser utilizado para a análise dos demais diagramas do código.

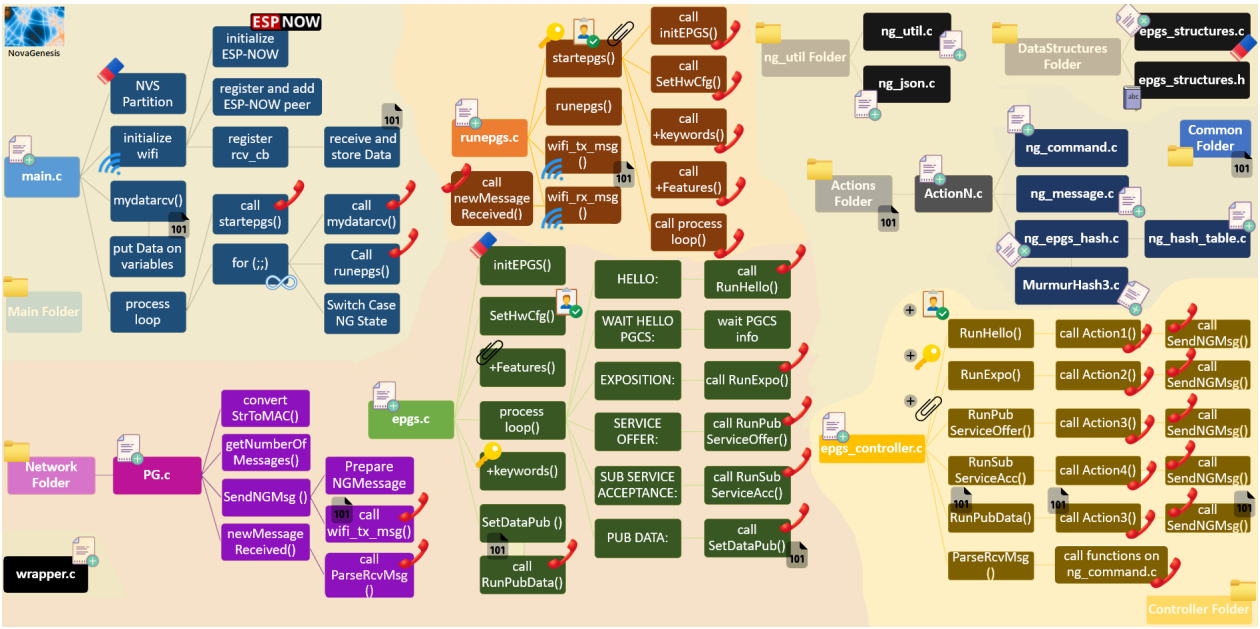

Figura 4.12: Diagrama de programação do microcontrolador ESP32-02.

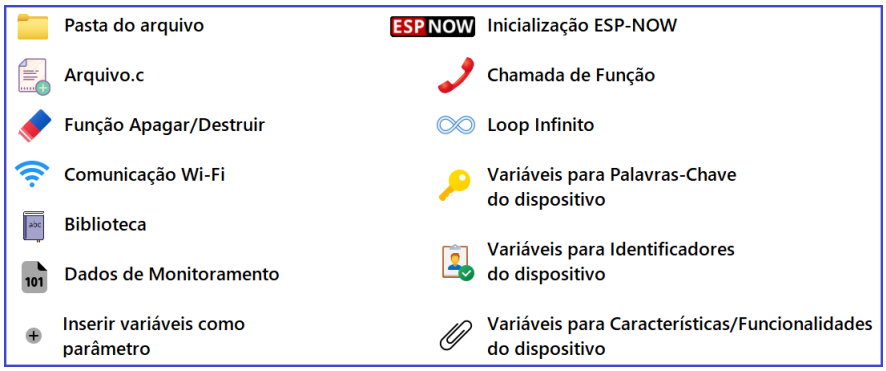

Figura 4.13: Mapeamento de funcionalidades do Diagrama de programação do microcontrolador ESP32-02.

O diagrama do código do microcontrolador ESP32-02 consiste de módulos e arquivos fonte que permitem o funcionamento simultâneo de envio/recepção de dados via comunicação ESP-NOW e da NG embarcada. O EPGS foi desenvolvido por Vaner Magalhães [77] a partir do PGCS com o intuito de criar a NG embarcada em um microcontrolador NXP LPC1769 [121]. Além disso, Gabriel Scarpioni focou seus estudos para utilizar o *software* embarcado em uma aplicação específica de IoT envolvendo um sensor para coleta de dados do mesmo [122]. Baseado nestes trabalhos anteriores, os principais desafios para esta dissertação foram a adaptação da NG para sua versão embarcada no microcontrolador ESP32, conciliando o envio de mensagens NG e comunicação ESP-NOW de forma simultânea, além de atender as camadas propostas pelo modelo RAMI 4.0. Para o desenvolvimento dessa aplicação, foram necessárias alterações no código para que o programa fosse executado em um novo microcontrolador, que é o ESP32, devido à seu baixo custo comparado ao NXP LPC1769, além de ser capaz de atender a todos os requisitos necessários da aplicação, com a vantagem de uso da comunicação ESP-NOW para se comunicar com outros microcontroladores diretamente pela camada Wi-Fi. Além disso, foram realizadas alterações nos arquivos *main.c*, *runepgs.c*, *epgs.c*, *epgs_controller* e *PG.c*, além das bibliotecas, de modo que o projeto pudesse atender as expectativas em um cenário de Indústria 4.0 com um modelo ICPS baseado no RAMI 4.0 para a digitalização e monitoramento de múltiplos equipamentos conectados a um PLC de forma simultânea através dos dados recebidos pela comunicação ESP-NOW. A seguir, serão descritos os códigos dos arquivos NG de forma organizada, dividindo a Figura 4.12 de acordo com o módulo discutido, e detalhando principalmente os arquivos mencionados acima que foram modificados para o funcionamento da aplicação.

Main Folder:

O *Main Folder* consiste da pasta responsável por englobar o arquivo principal, denominado *main.c*. Este arquivo é responsável por iniciar a execução do programa EPGS e seu código está disponível no Apêndice B.1. A Figura 4.14 destaca o arquivo *main.c* e suas principais funcionalidades, que serão descritas a seguir.

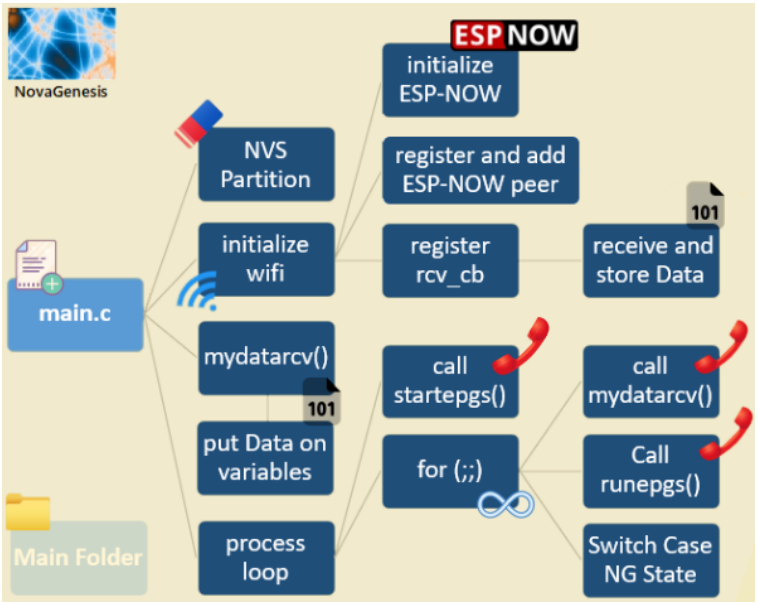

Figura 4.14: Diagrama do arquivo main.c do EPGS NG.

1. **NVS Partition:** Esta seção do código é responsável pela partição do Non-Volatile Storage (NVS), em português, Armazenamento Não-Volátil, que usa uma porção da memória flash. Por isso, é comum primeiramente apagar os conteúdos da partição NVS padrão para depois inicializá-la;

2. **initialize wifi:** Esta seção chama a função de inicialização do Wi-Fi, configurando todos os parâmetros necessários para em seguida conectar a uma rede através do SSID e Password pré-definidos. Dentre as configurações estabelecidas, é preciso definir o dispositivo como Modo Station. Por padrão, a tarefa Wi-Fi do ESP32 funciona no modo Modem-sleep para economia de energia, e quando há uma quantidade grande de dados sendo recebidos via Wi-Fi em um curto período, pacotes podem ser perdidos. Para esta aplicação, foi necessário desabilitar o modo de consumo de energia para que os pacotes transmitidos pelo ESP32-01 sejam recebidos pelo ESP32-02 sem perdas significativas. Além da inicialização do Wi-Fi, esta seção provê a inicialização da comunicação ESP-NOW, seguido pelo registro e adição do par ESP-NOW e registro da *callback* de recepção. O registro e adição do par ESP-NOW refere-se às informações do endereço MAC do ESP32-01 e do canal de comunicação usado para permitir troca de dados, que é definido como "1" conforme já detalhado na Parte 2.1 do Cenário Geral. Já o registro da *callback* de recepção tem como objetivo receber os dados do *buffer* de armazenamento dos estados das saídas digitais %Q0.0, %Q0.1 e %Q0.2 do PLC enviadas por Registradores Holding ao ESP32-01 e armazená-los na struct *HR_Data*. Essa struct é composta por 3 variáveis do tipo inteiro (*HR_0*, *HR_1* e *HR_2*) para alocar o estado das 3 saídas digitais a serem monitoradas;

3. **mydatarcv():** Refere-se à função *my_data_receive* para o armazenamento dos dados em variáveis separadas. Para isso, ela recebe como parâmetro a struct *HR_Data* mencionada anteriormente, e armazena seus valores (*HR_Data->HR_0*, *HR_Data->HR_1* *HR_Data->HR_2*) recepectivamente nas variáveis *Hold_Reg_0*, *Hold_Reg_1* e *Hold_Reg_2*.

4. **process loop:** Refere-se à tarefa NG criada na função principal. Devido ao microcontrolador ESP32 possuir 2 cores de processamento, essa task foi definida para ser executada no core 1. O motivo para isso é separar o uso de processamento do mesmo, visto que as tasks Wi-Fi são configuradas por padrão para serem executadas no core 0, gerenciando melhor o uso dos dois processadores do ESP32 e suas respectivas CPUs. É importante destacar que, apesar da possibilidade de configurar as task Wi-Fi para serem executadas no core 1, tasks de eventos não podem ser movidas, e neste caso os eventos Wi-Fi continuariam sendo despachados para o core 0. Basicamente, essa tarefa é responsável por chamar a função *startepgs()* do arquivo *runepgs.c* e iniciar um loop infinito, que consiste em chamar a função *my_data_receive* previamente descrita, a função *runepgs()* do arquivo *runepgs.c* e monitorar o ciclo de vida NG, apresentando o estado NG atual.

Os demais arquivos do código estão localizados no módulo **components/epgs** e serão detalhados a seguir.

runepgs.c:

Basicamente, a função do arquivo *runepgs.c* é receber as chamadas de funções do arquivo *main.c*, definir os NBs da aplicação, identificadores, palavras-chave e características do dispositivo industrial digitalizado, semelhante ao I4.0C descrito no RAMI 4.0 na Seção 2.2. Além disso, este arquivo realiza as configurações necessárias para a execução do EPGS através de chamadas das funções disponíveis no arquivo *epgs.c* para armazenar as informações do dispositivo, além da função responsável por enviar dados pela rede Wi-Fi. O código completo do arquivo *runepgs.c* está disponível no Apêndice B.2, e a Figura 4.15 enfatiza o arquivo e suas principais atividades, que são descritas em seguida.

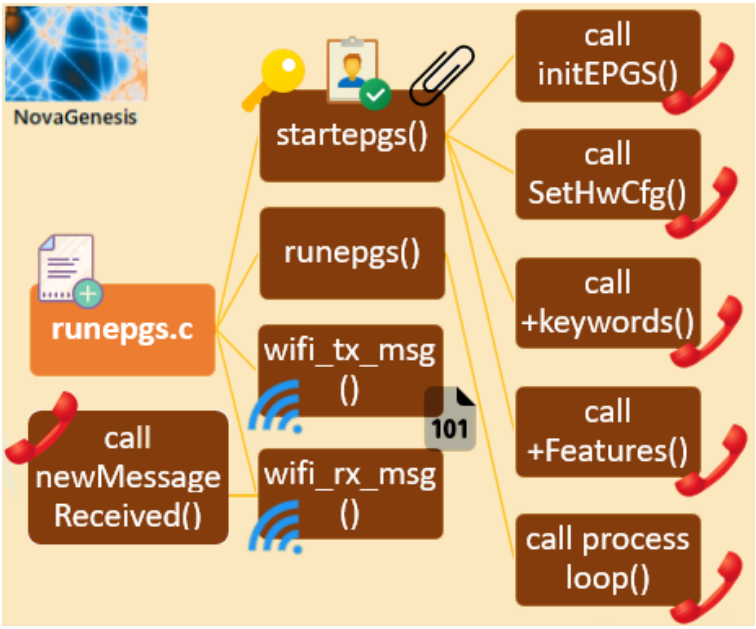

Figura 4.15: Diagrama do arquivo runepgs.c.

1. **startepgs():** Refere-se à função responsável por publicar nomes para definir os identificadores do microcontrolador e características do dispositivo industrial (PLC e suas respectivas saídas) e encaminhá-las para o arquivo *epgs.c*. Desta forma, a Tabela 4.2 descreve os principais identificadores, características e funcionalidades do dispositivo:

Tabela 4.2: Identificadores, características e funcionalidades do dispositivo.

Variável	Definição	Valor Atribuído
ucIdentify	Identificador (MAC) do microcontrolador	24:6F:28:AA:B9:68
ucHID	Identificador do *Hardware*	12345
ucSOID	Identificador do Sistema Operational	NG_SO
ucPID	Identificador do Processo	4321
ucStack	Pilha de rede	Wi-Fi
ucInterface	Interface de rede	eth1
ucPLCDeviceType	Tipo de dispositivos do PLC monitorados	OutputDevices

Desta forma, esta estrutura implementada é capaz de criar identificadores únicos não só para o dispositivo a ser digitalizado, mas também para as demais configurações da rede, de forma fácil e flexível, podendo ser alterada de acordo com as necessidades do usuário e mudanças na aplicação. Além disso, é válido ressaltar a possibilidade de uso de nomes em linguagem natural, trazendo mais clareza ao usuário e criando uma iteração mais próxima à máquina, assim como é conceituado na Indústria 5.0.

Realizadas as declarações necessárias, as ações são tomadas de acordo com a Figura

4.15, através de chamadas de funções presentes no arquivo *epgs.c*:

- **call InitEPGS()**: Chamada da função *InitEPGS()*;
- **call SetHwCfg()**: Chamada da função *setHwConfigurations*, enviando como parâmetros *ucHID, ucSOID, ucStack, ucInterface e ucIdentify*;
- **call +keywords()**: Chamada da função *addKeyWords*, enviando como parâmetros *ucPLCDeviceType*;
- **call +Features()**: Chamada da função *addHwSensorFeature* para enviar como parâmetros NBs correspondentes às características e funcionalidades do dispositivo a ser monitorado. A Tabela 4.3 descreve os respectivos nomes e valores inseridos.

Tabela 4.3: Funcionalidades do dispositivo industrial: Nomes e Valores.

Nome	**Valor**
PLCDeviceType	OutputDevices
MotorOutput	Q0
PistonOutput	Q1
LampOutput	Q2
OutputON	1
OutputOFF	0

Como esta aplicação propõe a simulação de monitaremento de *buffer* de armazenamento do estado de 3 saídas digitais do PLC (ex.: Motor, Pistão e Lâmpada), essas definições permitem ao PGCS reconhecer os tipos de equipamentos sendo monitorados, sua descrição e suas funcionalidades para a operação, assim como é exigido nas camadas de arquitetura do modelo de referência RAMI 4.0;

2. **runepgs():** Chamada da função *processloop*;
3. **wifi_tx_msg() e wifi_rx_msg():** Funções para, respectivamente, transmissão e recepção de mensagens via Wi-Fi, incluindo dados de monitoramento e controle. Em *wifi_rx_msg()*, é chamada a função *NewMessageReceived* do arquivo *PG.c*.

Além do arquivo fonte, há também a biblioteca *runepgs.h* que contém declarações de variáveis usadas em outros arquivos para o funcionamento do projeto. Ela pode ser vista no Apêndice B.3.

epgs.c:

O arquivo *epgs.c* tem como objetivo atender às chamadas de funções realizadas no arquivo *runepgs.c* e executá-las com os parâmetros recebidos, armazenando todas as informações na struct NgEPGS, responsável por receber e organizar todas as variáveis do processo de comunicação NG entre ESP32-02 e PGCS. O código completo deste arquivo está disponível no Apêndice B.4 e a Figura 4.16, ilustra suas principais funções, definidas a seguir.

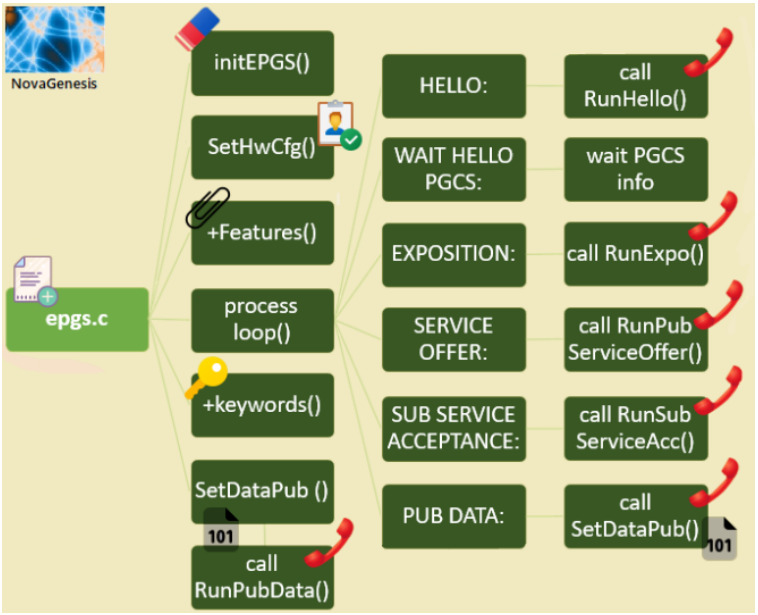

Figura 4.16: Diagrama do arquivo epgs.c.

1. **initEPGS():** Função de inicialização para zerar contadores e apagar valores das variáveis da struct *NgEPGS*. Além disso, são criadas palavras-chave NG responsáveis por representar o dispositivo, com os nomes "EPGS" e "Embedded_Proxy_Gateway_Service";

2. **SetHwCfg():** Refere-se à função *setHwConfigurations* para atribuir os parâmetros de informações da rede recebidos da chamada (Tabela 4.2) em suas respectivas variáveis da struct *NgEPGS*:

   NgEPGS->NetInfo->SOID,

   NgEPGS->NetInfo->HID,

   NgEPGS->NetInfo->Stack,

   NgEPGS->NetInfo->Interface;

   NgEPGS->NetInfo->Identifier;

3. **+keywords():** Refere-se à função *addKeyWords* para atribuir os valores das palavras-chave criadas para a representação do dispositivo (Tabela 4.2 e função *initEPGS()*) na variável:

   NgEPGS->HwDescriptor->keyWords;

4. **+Features():** Refere-se à função *addHwSensorFeature* para inserir os nomes e valores referentes às características dos dispositivos recebidos da chamada (Tabela 4.3) nas respectivas variáveis:

   NgEPGS->HwDescriptor->SensorFeatureName;

NgEPGS->HwDescriptor->SensorFeatureValue;

5. **processloop():** Essa função recebe como parâmetro a struct *NgEPGS* e inicia um loop do Switch Case baseado na variável *NgEPGS->State*, que recebe o estado atual da comunicação NG. Os estados e suas respectivas ações são descritas a seguir:

   - **HELLO:** Chama a função *RunHello* do arquivo *epgs_controller.c* e inicia um contador para que após um número determinado de passagens pelo case (ex.: 5), o estado seja alterado para WAIT_HELLO_PGCS;
   - **WAIT_HELLO_PGCS:** Estado de espera até o dispositivo receber as informações de rede e identificadores do PGCS. Este estado é importante para realizar o pareamento entre os dois dispositivos, de modo que o microcontrolador reconheça o computador de monitoramento remoto para a comunicação. Quando recebidos, *NgEPGS->State* recebe o estado ESPOSITION;
   - **EXPOSITION:** Chama a função *RunExposition* do arquivo *epgs_controller.c* e muda o estado para SERVICE_OFFER, iniciando um contador. Esta função envia as palavras-chave do dispositivo, permitindo uma resolução de nomes ilimitada, além da possibilidade de transmissão de dados técnicos e funcionais relacionados ao dispositivo a ser digitalizado;
   - **SERVICE_OFFER:** Chama a função *RunPubServiceOffer* do arquivo *epgs_controller.c* caso o contador seja menor que um número previamente determinado (ex.: 5). Feito isso, o estado é alterado para WAIT_SERVICE_ACCEPTANCE_NOTIFY e o contador é incrementado. WAIT_SERVICE_ACCEPTANCE_NOTIFY retorna ao estado SERVICE_OFFER até o contador atingir o valor esperado, para finalmente *NgEPGS->State* receber o próximo estado. Este contador tem como objetivo garantir que a função *RunPubServiceOffer* seja chamada antes de iniciar o próximo estado;
   - **SUBSCRIBE_SERVICE_ACCEPTANCE:** Chama a função *RunSubscribeServiceAcceptance* caso o contador seja menor que um número previamente determinado (ex.: 10). Feito isso, o estado é alterado para WAIT_SERVICE_ACCEPTANCE_DELIVERY e o contador é incrementado. WAIT_SERVICE_ACCEPTANCE_DELIVERY retorna ao estado SUBSCRIBE_SERVICE_ACCEPTANCE até o contador atingir o valor esperado, para finalmente *NgEPGS->State* receber o próximo estado. Similar ao estado anterior, o objetivo do contador é garantir a chamada e execução da função *RunSubscribeServiceAcceptance*;
   - **PUB_DATA:** Para o monitoramento de dados, este estado utiliza uma função para armazenar em um *buffer* denominado *ucFile* uma *string* contendo o valor das 3 variáveis criadas no arquivo *main.c* que guardam os valores dos estados das saídas digitais do PLC (*Hold_Reg_0*, *Hold_Reg_1* e *Hold_Reg_2*) armazenadas no *buffer*. Também é criado um *buffer* denominado *ucBufferName* para inserir o nome do arquivo cuja *string* de dados será inserida, definido com extensão ".json" para esta aplicação. Então, a função *setDataToPub* é chamada com os parâmetros

dos *buffers* passados e o contador é incrementado para que o *Switch Case* não possa entrar nos estados anteriores, criando a partir daí um *loop* em PUB_DATA.

6. **setDataPub():** Refere-se à função SetDataToPub, que recebe os *buffers* com as strings do nome do arquivo (*ucBufferName*) e dos dados das saídas digitais do PLC (*ucFile*) e os armazena, respectivamente, nas seguintes variáveis da struct *NgEPGS:*

   NgEPGS->pubDataFileName;

   NgEPGS->pubData;

   Feito isso, a função *RunPublishData* do arquivo *epgs_controller.c* é chamada;

Além do arquivo fonte, há também a biblioteca *epgs.h* que contém declarações de variáveis usadas em outros arquivos para o funcionamento do projeto. Ela pode ser vista no Apêndice B.5.

Controller Folder:

O módulo Controller é composto pelos arquivos *epgs_controller.c* e *epgs_controller.h*. O arquivo *epgs_controller.c*, cujo código completo pode ser visto no Apêndice B.6, executa funções específicas de cada estado do ciclo de vida NG que foi chamado no código *epgs.c* descrito anteriormente. Já a biblioteca *epgs_controller.h* contém a declaração das funções usadas no arquivo *epgs_controller.c* e pode ser observada no Apêndice B.7. A Figura 4.17 ilustra as principais ações do arquivo controlador *epgs_controller.c* e é detalhado a seguir.

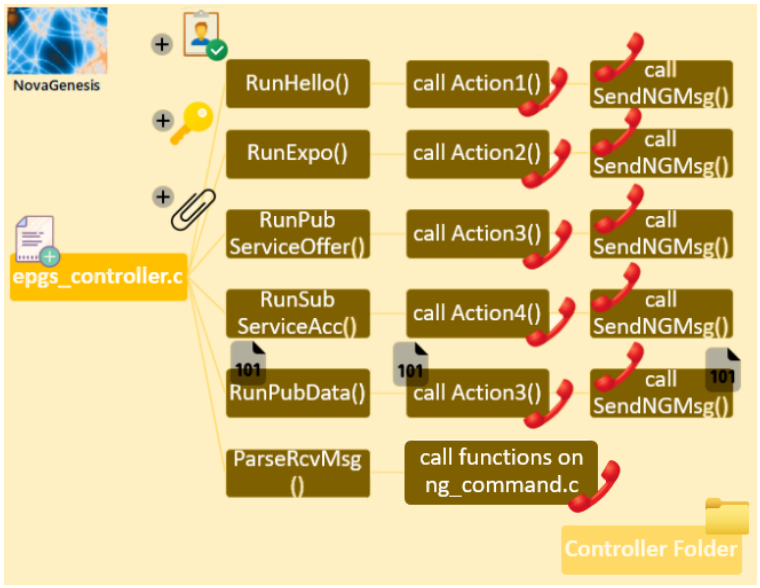

Figura 4.17: Diagrama do arquivo epgs_controller.c.

1. **RunHello():** Refere-se à função *RunHello* chamada no arquivo *epgs.c* quando o estado da NG está em HELLO. Esta função tem como objetivo chamar a função *ActionRunHello*, definida como *Action1()* na Figura 4.17 e localizada no módulo *Action Folder*, como mostrado na Figura 4.12. Os parâmetros passados são *ngEPGS->NetInfo* com as informações da rede definidas na Tabela 4.2 e uma mensagem NG criada e denominada *HelloMessage*. Feito isso, a função *SengNGMessage* disponível no arquivo *PG.c* é chamada, enviando como parâmetros: a struct *NgEPGS*; a mensagem *HelloMessage* após ser tratada pela *Action1()*; e a variável *booleana isBroadcast* como "true";

2. **RunExpo():** Refere-se à função *RunExposition* chamada no arquivo *epgs.c* quando o estado da NG está em EXPOSITION. Seu principal objetivo é chamar a função *ActionExposition*, definida como *Action2()* na Figura 4.17 e localizada no módulo *Action Folder* ilustrado na Figura 4.12. Os parâmetros passados são a struct *ngEPGS* para enviar as palavras-chave criadas e uma mensagem NG criada e denominada *expositionMessage*. Feito isso, a função *SengNGMessage* disponível no arquivo *PG.c* é chamada, enviando como parâmetros: a struct *NgEPGS*; a mensagem *expositionMessage* após ser tratada pela *Action2()*; e a variável *booleana isBroadcast* como "false";

3. **RunPubServiceOffer():** Refere-se à função *RunPubServiceOffer* chamada no arquivo *epgs.c* quando o estado da NG está em SERVICE_OFFER. O ciclo de vida com estabelecimento de contratos foi desenhado pelo professor Antônio Marcos Alberti para ser implementado no PGCS, e posteriormente adaptado para o EPGS por Gabriel Scarpioni [122] e Vaner Magalhães [121], conforme já mencionado anteriormente. Primeiramente, é criada uma variável chamada *FilePayload* cujo conteúdo consiste em uma *string* com linhas de comando NG para publicações de NBs. São adicionadas a essas Linhas de Comando uma chave de contrato denominada *EPGS_PLC*, juntamente das características e funcionalidades do dispositivo industrial definidas na Tabela 4.3, como nomes e valores armazenados nas variáveis *NgEPGS->HwDescriptor->sensorFeatureName* e *NgEPGS->HwDescriptor->sensorFeatureValue*, conforme já mostrado anteriormente. Então, é chamada a função *actionPublicationAndNotification*, definida como *Action3()* na Figura 4.17 e localizada no módulo *Action Folder*, como mostrado na Figura 4.12. Os parâmetros passados são: a struct *ngEPGS*; a variável *booleana IsData* como "false", já que os dados monitorados não são encapsulados nesta etapa; a variável *FileName* indicando o nome do documento a ser publicado, que nesse caso é a oferta de serviço declarada como "Service_Offer.txt"; a variável *FilePayload* referente ao *payload* inserido no documento criado, cujo conteúdo foi previamente descrito; e uma mensagem NG criada e denominada *pubServiceOfferMessage*. Feito isso, a função *SengNGMessage* disponível no arquivo *PG.c* é chamada, enviando como parâmetros a struct *NgEPGS*, a mensagem *pubServiceOfferMessage* após ser tratada pela *Action3()*, e a variável *booleana isBroadcast* como "false";

4. **RunSubServiceAcc():** Refere-se à função *RunSubscribeServiceAcceptance* chamada no arquivo *epgs.c* quando o estado da NG está em SUBSCRIBE_SERVICE_ACCEPTANCE. Esta função tem como objetivo chamar a função *ActionSubscriptionServiceAccep-*

tance, definida como *Action4()* na Figura 4.17 e localizada no módulo *Action Folder*, como mostrado na Figura 4.12. Os parâmetros passados são: *NgEPGS->NetInfo* com as informações da rede; informações de identificadores do PGCS obtidas após as trocas de mensagens entre o par NG nos estados anteriores; um contador de mensagens; chave de contrato recebida pelo PGCS; e uma mensagem NG criada e denominada *subServiceAcceptanceMessage*. Feito isso, a função *SengNGMessage* disponível no arquivo *PG.c* é chamada, enviando como parâmetros a struct *NgEPGS*, a mensagem *subServiceAcceptanceMessage* após ser tratada pela *Action4()*, e a variável *booleana isBroadcast* como "false";

5. **RunPubData():** Refere-se à função *RunPublishData* chamada no arquivo *epgs.c* quando o estado da NG está em PUB_DATA. Sua funcionalidade é chamar a função *actionPublicationAndNotification*, definida como *Action3()* na Figura 4.17 e localizada no módulo *Action Folder*, como mostrado na Figura 4.12. Apesar de ser a mesma função chamada em *RunPubServiceOffer*, os seguintes parâmetros passados diferem: a variável booleana *IsData* é declarada como "true", já que os dados monitorados serão encapsulados nesta etapa; a variável *FileName* recebe o valor de *NgEPGS->pubDataFileName*, declarado no arquivo *epgs.c*, referente ao nome do documento gerado para inserir os dados das saídas digitais do PLC; a variável *FilePayload* recebe o valor de *NgEPGS->pubData*, também declarado no arquivo *epgs.c*, e que contém um *buffer* com uma *string* dos dados das saídas digitais do PLC; e uma mensagem NG criada e denominada *pubDataMessage*. Feito isso, a função *SendNGMessage* disponível no arquivo *PG.c* é chamada, enviando como parâmetros a struct *NgEPGS*, a mensagem *pubDataMessage* após ser tratada pela *Action4()*, e a variável *booleana isBroadcast* como "false";

6. **ParseRcvMsg():** Refere-se à função *ParseReceivedMessage* para analisar as mensagens recebidas. Para isso, são necessárias chamadas de funções do arquivo *ng_command.c* responsáveis por analisar linhas de comando de mensagens. Uma das características desta função é avaliar o *payload* recebido através das linhas de comando das mensagens NG para poder coletar informações referentes a identificadores do par.

Action e Common Folders:

O módulo Action é composto pelos arquivos responsáveis por compôr as Actions chamadas no arquivo *epgs_controller.c* previamente descrito. No diagrama da Figura 4.12, esses arquivos são definidos como *ActionN.c*, referente à Action chamada no controlador. É válido ressaltar que esta estrutura faz parte da arquitetura NG que é executada no computado de monitoramento e controle remoto, e foi adaptada por Gabriel Scarpioni [122] e Vaner Magalhães [121] para ser executada na versão embarcada da mesma forma. O objetivo de cada uma dessas Actions é criar a mensagem NG requisitada de acordo com o estado atual. Como exemplo, no estado HELLO, o arquivo *epgs_controller.c* executa a função *RunHello*, que cria uma mensagem denominada *HelloMessage* e chama a função *Action1*. Então, o objetivo da *Action1* é criar as linhas de comando necessárias para a construção da mensagem de Hello,

para então retorná-la ao controlador e finalmente enviar a mensagem completa ao arquivo *PG.c* através da função *SendNGMessage()*. Este passo-a-passo é o mesmo para as demais Actions, variando as Linhas de Comando Específicas de acordo com a mensagem a ser construída. É válido lembrar que a construção das Linhas de Comando dessas mensagens foi detalhada na Seção 2.3.1, através das Figuras 2.10 e 2.11.

Para criar essas Actions, são chamadas funções pertencentes a arquivos do módulo Common. Esses arquivos, por sua vez, são:

1. **ng_command.c:** Arquivo com funções para criar e analisar uma Linha de Comando e sua estrutura em geral;
2. **ng_message.c:** Arquivo com funções para criar Mensagens NG com as Linhas de Comando;
3. **ng_epgs_hash.c:** Arquivo com funções para gerar um código hash com os SVNs dos nomes requisitados. Para isso, faz uso do arquivo *MurmurHash3.c*;
4. **MurmurHash3.c:** Arquivo com algoritmo do MurmurHash3 para gerar os SVNs dos nomes solicitados;
5. **ng_hash_table.c:** Arquivo com funções relacionadas a Tabela Hash (HT), como por exemplo, criar uma tabela hash e pegar ou armazenar determinado NB ou conteúdo através de seus nomes e valores previamente definidos.

A Figura 4.18 ilustra os módulos Action e Common.

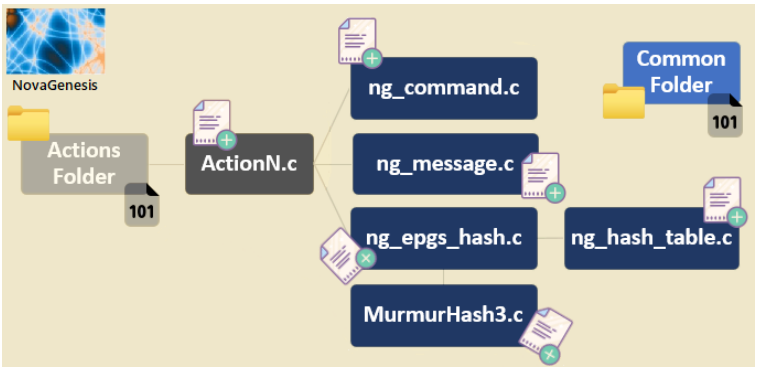

Figura 4.18: Diagrama dos módulos Action e Common.

Network Folder:

O módulo Network é responsável pelo Proxy/Gateway da arquitetura NG através do arquivo *PG.c*. A Figura 4.19 ilustra o Folder Network com o arquivo *PG.c* e suas respectivas ações descritas a seguir:

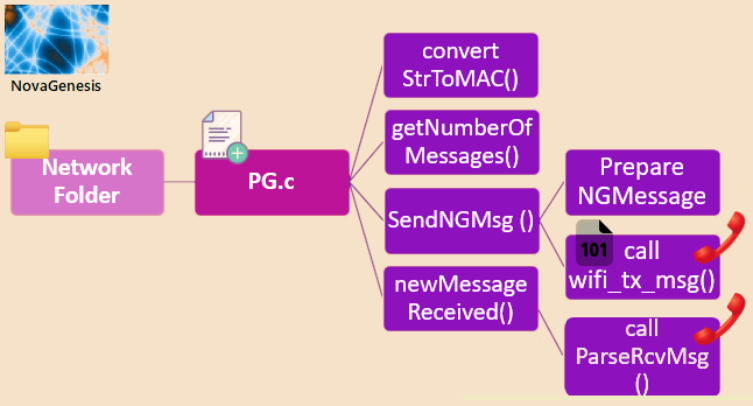

Figura 4.19: Diagrama do módulo Network.

1. **convertStrToMAC():** Função para converter uma string MAC (11:22:33:44:55:66 com 17 caracteres) em um vetor de 6 bytes;

2. **getNumberOfMessages():** Função para calcular o número de mensagens recebidas pelo par NG;

3. **SendNGMsg():** Refere-se à função *SendNGMessage* chamada no pelo arquivo *epgs_controller.c*. Essa função recebe como parâmetros: a struct *NgEPGS* com informações, por exemplo, dos identificadores dos pares NG; uma variável denominada *Message*, referente à mensagem que foi passada na chamada, de acordo com o atual ciclo de vida NG; e a variável *isBroadcast* para definir o endereço de destino MAC no qual a mensagem será enviada. Então, a mensagem é preparada de acordo com a tecnologia a ser aplicada para sua transmissão, como por exemplo Wi-Fi nesta aplicação, e enfim a função *wifi_tx_msg* do arquivo *runepgs.c* é chamada para que o pacote possa ser transmitido.

4. **NewMessageReceived():** É chamada pela função *wifi_rx_msg* do arquivo *runepgs.c* como modo de sinalização de nova mensagem recebida pelo par NG via Wi-Fi. Esta função realiza o encaminhamento da mensagem NG para a função *ParseReceivedMessage* do arquivo *epgs_controller.c* afim da mesma ser analisada.

ng_util Folder, DataStructure Folder e arquivo wrapper.c:

O módulo ng_util contém os arquivos *ng_util.c*, *ng_json.c* e suas respectivas bibliotecas. O arquivo *ng_util* é composto pela função C++ **strtok_r**, que consiste em dividir uma *string* por um delimitador, e é usada em funções do código, como por exemplo, na função *convertStrToMAC* do arquivo *PG.c*, onde a string do endereço MAC é separada através do delimitador ":", ajudando em seu processo de conversão para um vetor de 6 bytes. Já o arquivo *ng_json.c* consiste em um conjunto de funções para criar um arquivo ".json" e permitir a inserção de dados nele.

O módulo DataStructure é composto pelos arquivos *DataStructure.c*, constituído por um conjunto de funções para apagar as mensagens, identificadores do EPGS e do PGCS e demais informações da struct *NgEPGCS* quando chamadas; e a biblioteca *DataStructure.h*, responsável por declarar as principais variáveis executadas por todo o código, como por exemplo a própria struct *NgEPGS*. Por esse motivo, essa biblioteca é chamada nos demais arquivos descritos anteriormente.

A Figura 4.20 mostra os módulos *ng_util* e *DataStructure*.

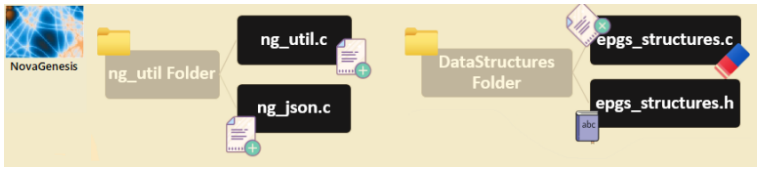

Figura 4.20: Diagrama dos módulos ng_util e DataStructure.

Por fim, o arquivo *wrapper.c*, tem como objetivo o encapsulamento de funções comuns em C++ adaptadas ao formato NG para sua chamada nos demais arquivos do código. Como exemplo, a função *int ng_atoi (const char* str)*, na verdade, retorna a função *atoi(str)*, que em linguagem C++ converte uma variável *string* para inteiro. Além da extensão ".c", também há a biblioteca *wrapper.h*, cujas funções são declaradas para serem usadas nos outros arquivos do projeto.

Para que o EPGS do microcontrolador ESP32-02 se comunique com o computador de monitoramento remoto, é necessário que o mesmo execute o arquivo *IoTTestApp* para parear com o dispositivo e receber as mensagens NG desejadas. O arquivo *IoTTestApp*, por sua vez, é um código que foi desenvolvido pelo prof. Antônio Marcos Alberti, executado no *core* da NG com o intuito de receber dados de monitoramento. Os estados da NG mencionados anteriormente no EPGS são executados e o recebimento das mensagens NG de publicação de dados na extensão ".json"são organizados em um arquivo ".txt"que é atualizado em tempo real à medida que os dados chegam, permitindo o monitoramento remoto dos ativos digitalizados.

Capítulo 5

Resultados e Análises

Neste Capítulo serão apresentadas a Metodologia e Procedimento de Testes, seguidos pelos resultados e análise obtidos a partir do Cenário Geral descrito no Capítulo 4, de modo a validar a viabilidade da arquitetura NG como proposta ao modelo RAMI 4.0.

5.1 Metodologia de Testes

O intuito da metodologia de testes consiste em detalhar o que será feito para validar a NG embarcada como uma arquitetura de Internet do Futuro que possa ser usada para desenvolver um ICPS baseado no modelo RAMI 4.0, com a vantagem dentre as demais arquiteturas de se preocupar com a emestão de IIoT em I4.0 e com as limitações da Internet atual.

Primeiramente, será implementado um Procedimento de Testes com a Parte 1 do Cenário Geral descrito no Capítulo 4, cujo objetivo é criar um ambiente industrial real com ativos e protocolos legados comumente usados em fábrica para posteriormente realizar os testes do ICPS a ser desenvolvido com a arquitetura NG. Porém, é importante ressaltar que, devido às circunstâncias já apresentadas no capítulo anterior, a validação prática da viabilidade do ICPS será realizada através do ambiente real com o *buffer* de armazenamento de estados das saídas digitais do PLC simulado via *software* Modbus Poll. Portanto, o objetivo deste primeiro Procedimento de Testes é primeiramente mostrar como é fisicamente o cenário real em que o ICPS será testado, de depois detalhar os testes a serem feitos neste cenário simulado.

Feito isso, será implementado um Procedimento de Testes com as Partes 2.1 e 2.2 do Cenário Geral. O objetivo para a Parte 2.1 é mostrar a eficiência do microcontrolador ESP32-01 na coleta de dados dos ativos em ambiente industrial. Já para a Parte 2.2, o foco é avaliar a viabilidade da arquitetura NG embarcada no microcontrolador ESP32-02 desenvolver um sistema ICPS baseado no modelo RAMI 4.0, entregando ao computador de monitoramento remoto os ativos digitalizados e seus respectivos dados, com o diferencial de lidar com as limitações da Internet atual discutidas ao longo desta dissertação.

Além disso, um Procedimento de Testes será criado para detalhar questões envolvendo o

uso de memória e modos de economia de energia do microcontrolador ESP32. Como o microcontrolador pode ser alimentado por uma fonte de alimentação de 5V, este valor energético de consumo torna-se insignificante em um ambiente fabril, já que o consumo de energia das indústrias é extremamente alto, vista a quantidade de equipamentos de alto valor energético em atividade ligados simultaneamente. Porém, o objetivo de discussão relacionada à economia de energia do microcontrolador é otimizar a solução, desenvolvendo um sistema que utilize a menor quantidade de recursos possíveis. Nesta etapa, serão realizados testes de execução de todo o sistema durante um período de 2 horas para transmitir 1200 pacotes de dados dos ativos, de forma a avaliar a taxa de pacotes entregues com sucesso para monitoramento. Estes testes serão feitos usando diferentes modos de economia de energia do microcontrolador ESP32 afim de avaliar o melhor cenário para o mesmo ser adotado.

Por fim, serão apresentadas as análises e os resultados obtidos a partir dos Procedimentos de testes descritos anteriormente.

5.2 Procedimentos de Testes para o Cenário Geral:

Esta Seção mostra os Procedimentos de Testes realizados para o desenvolvimento do ICPS em NG baseado no modelo RAMI 4.0 através do monitoramento remoto do *buffer* de armazenamento dos estados das saídas digitais de um PLC. Primeiramente, será apresentado o Procedimento de testes da Parte 1, responsável pela descrição de testes realizados para elaborar um ambiente industrial real, seguidos pelos Procedimentos de Testes das Partes 2.1 e 2.2, afim de detalhar os testes feitos para aquisição dos dados do ambiente fabril e digitalização de ativos através de uma arquitetura NG para o desenvolvimento de um ICPS. Por fim, será mostrado o Procedimento de Testes referentes ao consumo de memória e energia do microcontrolador ESP32.

5.2.1 Procedimento de Testes da Parte 1 do Cenário Geral

Para o Procedimento de Testes da Parte 1 do Cenário Geral, serão detalhados os testes em ambiente industrial físico da manipulação do *buffer* de armazenamento das saídas digitais do PLC a serem monitoradas. Assim sendo, o objetivo dos testes deste Procedimento foi simplesmente mostrar a eficiência do código Ladder desenvolvido em um ambiente físico para manipular as variáveis *booleanas* referentes às saídas digitais do PLC a serem armazenadas e monitoradas no *buffer*, de modo que as mesmas possam ser convertidas em valores inteiros para serem enviadas ao bloco de dados de transmissão do protocolo Modbus TCP/IP, validando sua representatividade para testar o ICPS a ser desenvolvido. Após a validação deste cenário para ser usado como teste para o desenvolvimento do ICPS, será apresentada a sua versão simulada com o *buffer* de armazenamento de estados das saídas digitais retratado no *software* Modbus Poll, já que foi adotada a avaliação desta solução baseada no simulador da Parte 1 do Cenário Geral para facilitar o gerenciamento de todo o sistema.

Em seguida, será apresentada a versão simulada do ambiente fabril real, com os equi-

pamentos de forma virtual, visto que este será o cenário em que a aplicação será avaliada posteriormente para realizar a aquisição dos dados dos ativos e enfim o desenvolvimento do ICPS.

A Figura 5.1 ilustra o PLC Siemens S7-1200 usado para os testes em seu estado inicial.

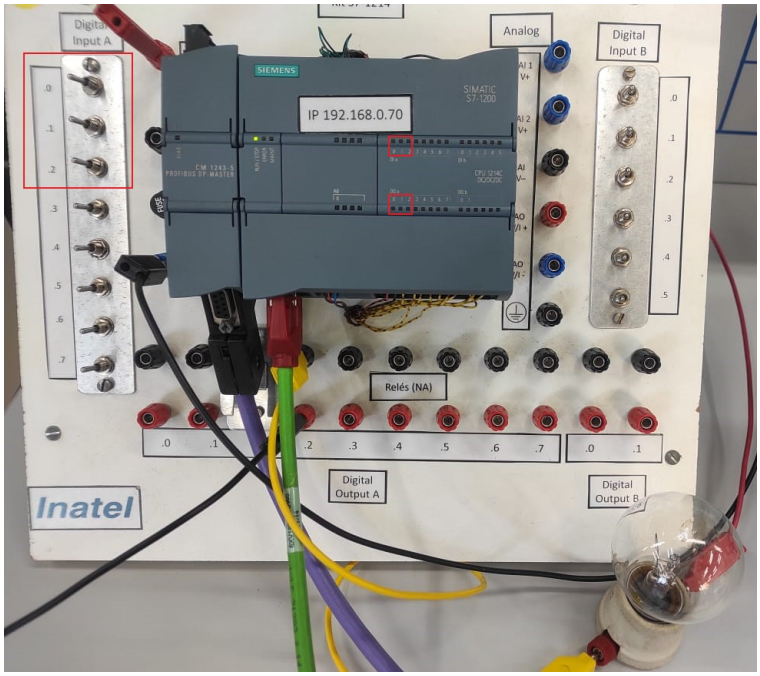

Figura 5.1: PLC no estado inicial da aplicação.

Como pode ser visto na Figura 5.1, as entradas digitais %I0.0, %I0.1 e %I0.2 estão conectadas às chaves ON/OFF à esquerda do painel. De acordo com o programa descrito na Seção 4.2, cada uma dessas entradas é responsável pelo acionamento das saídas %Q0.0, %Q0.1 e %Q0.2, respectivamente, e seus estados são armazenados em um *buffer* de monitoramento. Inicialmente, as chaves estão desligadas, e portanto as entradas digitais recebem o valor "0". Consequentemente, as saídas digitais também permanecem desacionadas. Além disso, foi conectada à %Q0.2 uma lâmpada, que está desligada devido ao nível lógico baixo da saída digital. O PLC S7-1200 permite monitorar o acionamento das entradas e saídas digitais através de *Light-Emitting Diodes* (LEDs), em português, Diodos Emissores de Luz, localizados na parte frontal de seu dispositivo. Como pode ser observado, tanto os LEDs sinalizadores das entradas quanto os LEDs da saída estão apagados, o que significa que os mesmos estão desligados.

A Figura 5.2 mostra as linhas de código do programa Ladder em seu estado inicial.

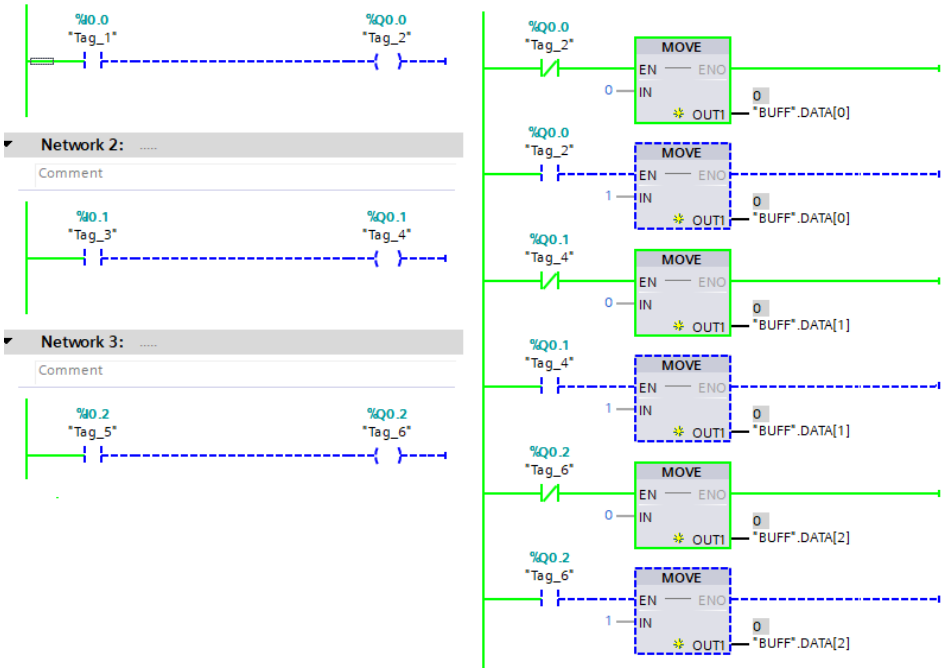

Figura 5.2: Linhas de Código do Programa Ladder em seu estado inicial.

Quando o PLC está executando o programa (modo RUN), as linhas do código ficam tracejadas em azul no caso de não haver conexão ou contato (0), e verdes no caso do contato estar sendo fechado e a conexão estabelecida (1). Conforme observado na Figura 5.2, percebe-se que no estado inicial, as entradas digitais estão desacionadas e consequentemente as saídas recebem o valor "0". Consequentemente, apenas os blocos *MOVE* cujos pinos *EN* estão conectados aos contatos normalmente fechados são acionados, enviando o valor inteiro "0" aos endereços *"BUFF".DATA[0]*, *"BUFF".DATA[1]* e *"BUFF".DATA[2]*. Estes valores são enviados no formato inteiro, pois não é possível utilizar o bloco *MOVE* para mover dados *booleanos*. Esse *buffer*, como mostrado na Seção 4.2, é o bloco de dados inserido no pino *MB_DATA_PTR* do bloco *MB_Client*, responsável pelos dados a serem transmitidos via protocolo Modbus TCP/IP. A Figura 5.3 ilustra o monitoramento do bloco de dados do *buffer* recebendo os valores "0" das saídas digitais em tempo real.

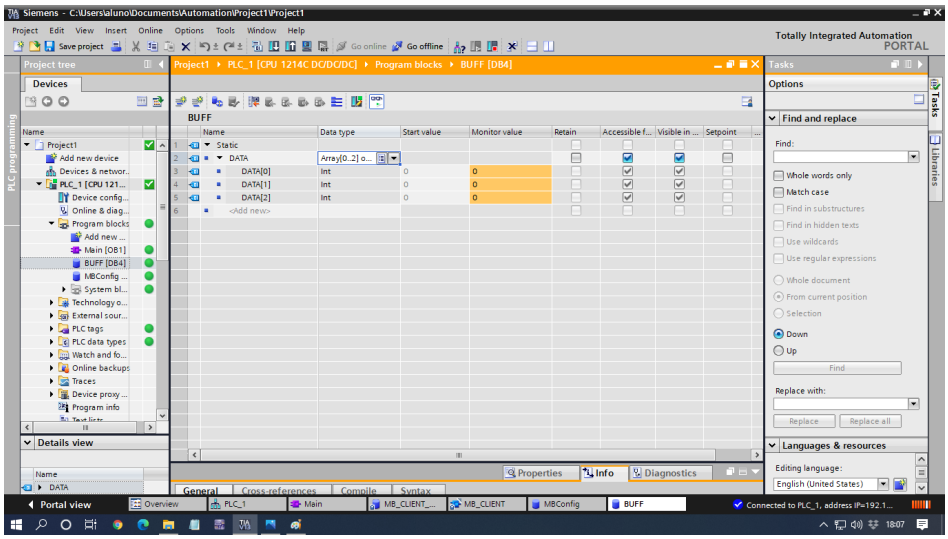

Figura 5.3: Bloco de Dados do *Buffer* Modbus TCP/IP com valores das variáveis iguais a 0.

Então, para testar o ambiente industrial criado, foi realizado o acionamento das entradas %I0.0 e %I0.2 através de suas respectivas chaves, cujo resultado esperado é o acionamento das saídas %Q0.0 e %Q0.2, que poderá ser visto no no TIA Portal V13, onde o programa do PLC é executado. Além disso, os dados referentes ao acionamento desses dispositivos devem ser enviados ao *buffer*, que conforme já detalhado no Capítulo 4.2 os transmite via protocolo Modbus ao ESP32-01. Estes resultados serão apresentados na Seção 5.3.1 referente aos resultados do Procedimento de Testes da Parte 1 do Cenário Geral.

Visto que a aquisição de dados dos ativos proviente do *buffer* de monitoramento do PLC foi feita a partir do simulador deste ambiente físico, é importante apresentar o *software* Modbus Poll, que foi usado para simular a TIS do PLC monitorando 3 saídas digitais. Assim sendo, para realizar a comunicação com o ESP32-01, basta inserir seu endereço IP como Modbus Servidor. A Figura 5.4 apresenta o programa mencionado.

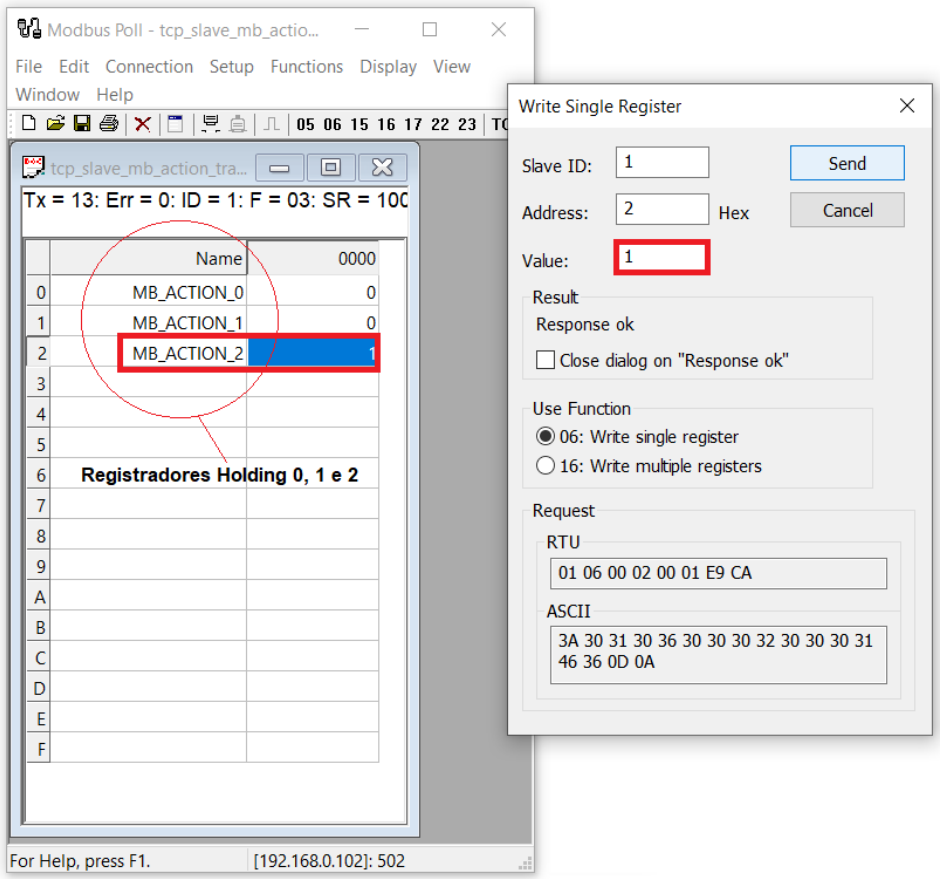

Figura 5.4: *Software* Modbus Poll usado para simular a TIS do PLC monitorando 3 saídas digitais.

Como pode ser visto na Figura 5.4, as variáveis nomeadas como "MB_ACTION_0", "MB_ACTION_1" e "MB_ACTION_2" representam os Registradores Holding 0, 1 e 2 do protocolo Modbus, e simulam, respectivamente, os dados provenientes das saídas %Q0.0 (motor), %Q0.1 (pistão) e %Q0.2 (lâmpada) sendo monitorados pelo *buffer* do PLC. Desta forma, ao alterar o valor das variáveis de forma manual, os respectivos registradores Holding das saídas também são alterados, enviando os dados atualizados via protocolo Modbus ao ESP32-02.

5.2.2 Procedimento de Testes da Parte 2.1 do Cenário Geral

O Procedimento de Testes da Parte 2.1 do Cenário Geral engloba a programação do microcontrolador ESP32-01, responsável pelo recebimento de dados do *buffer* de armazenamento dos estados das 3 saídas digitais do PLC via comunicação Modbus TCP/IP como servidor e envio dos dados ao ESP32-02 via comunicação ESP-NOW como transmissor. Conforme já informado no Capítulo anterior, a Parte 1 do Cenário Geral retrata apenas a manipulação de dados do PLC em ambiente físico para validar uma avaliação real de monitoramento via Modbus TCP/IP. Porém, para os testes de conexão e comunicação Modbus TCP/IP para

transmissão e recepção de dados, foi utilizado o *software* Modbus Poll responsável por simular o Ciente Modbus PLC S7-1200 e seu *buffer* de armazenamento de estado de suas saídas digitais através de seus respectivos Registradores Holding.

Desta forma, este Procedimento de Testes realiza a aquisição dos dados provenientes do *buffer* responsável por armazenar os estados das saídas digitais %Q0.0, %Q0.1 e %Q0.2 do PLC simuladas no Modbus Poll. Estes testes foram feitos usando o *software* Arduino IDE 1.8.13, configurando a porta COM5 para o microcontrolador, cujo objetivo é monitorar no terminal os dados recebidos via protocolo Modbus. O código é iniciado com a conexão à rede Wi-Fi e a informação do endereço IP do ESP32-01. A Figura 5.5 ilustra o terminal exibindo o endereço IP do dispositivo.

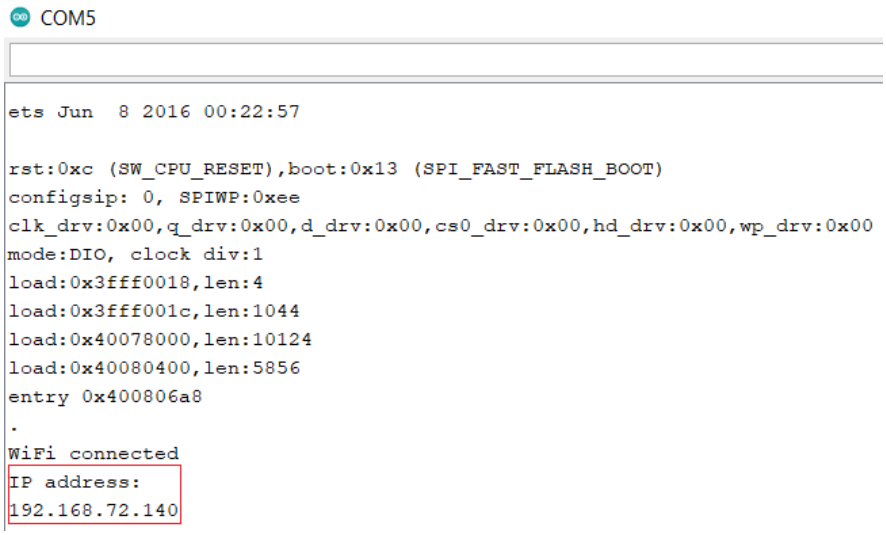

Figura 5.5: Terminal com exibição da conexão Wi-fi e endereço IP do ESP32-01, referente ao Servidor Modbus.

A informação do IP do microcontrolador é necessária para configurar no Cliente Modbus o endereço do Servidor. Com isso, o endereço IP coletado é inserido no *software* Modbus Poll para estabelecer a comunicação. Inicialmente, os estados das 3 saídas digitais foram definidos como ”0”. Além disso, foi criada uma variável no código do microcontrolador para contar a quantidade de pacotes entregues com sucesso ao ESP32-02 via comunicação ESP-NOW e calcular a sua porcentagem em tempo real baseada na quantidade total de pacotes de dados transmitidos. Por fim, foi criada uma função para exibir a quantidade de memória *heap* livre no microcontrolador. A Figura 5.6 ilustra o início de recepção e transmissão dos dados exibido no terminal monitor.

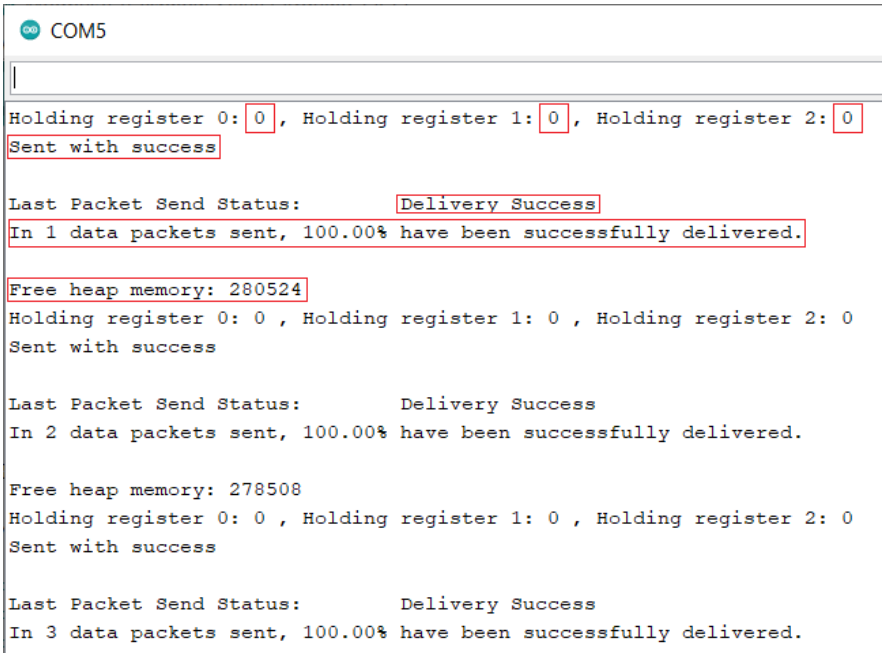

Figura 5.6: Terminal com exibição da recepção e transmissão de dados no ESP32-01.

Conforme visto pela Figura 5.6, o terminal monitor apresenta as seguintes informações destacadas:

- O *buffer* de armazenamento de dados com o estado das 3 saídas digitais do PLC simulado via *software* Modbus Poll sendo recebido via Modbus TCP/IP através dos Registradores Holding, inicialmente em "0";
- a sinalização de status de envio e entrega do pacote de dados ao ESP32-02 via comunicação ESP-NOW, indicando se foi transmitido com sucesso ou falha. O motivo dessas sinalizações se deve ao fato de que um pacote pode ser enviado corretamente e não ser entregue devido à uma falha na comunicação;
- uma mensagem indicando a quantidade de pacotes de dados enviados ao ESP32-02 até o momento, e a porcentagem de pacotes entregues com sucesso;
- a quantidade em bytes de memória heap livre no ESP32-01. A memória heap refere-se à porção de memória alocada dinamicamente durante a execução das instruções definidas pelo programador.

Para tornar o teste coerente com o Procedimento de Testes da Parte 1 do Cenário Geral, para a aquisição de dados do ambiente industrial, também foram simulados o acionamento das saídas %Q0.2 e %Q0.0, de modo a poder ser observado no terminal o recebimento destes dados via Modbus em tempo real. Estes resultados serão apresentados na Seção 5.3.2 referente aos Resultados do Procedimento de Testes da Parte 2.1 do Cenário Geral.

5.2.3 Procedimento de Testes da Parte 2.2 do Cenário Geral

Para esta Seção, foram realizados testes que permitem avaliar a arquitetura NG como um ICPS baseado no modelo RAMI 4.0. Para isso, o foco está no código do microcontrolador ESP32-02, programado na linguagem C++ no *software* Eclipse IDE 2021-03. Este programa realiza o monitoramento remoto dos ativos a partir dos dados das 3 saídas digitais do PLC provenientes de seu respectivo *buffer* de armazenamento simulado no *software* Modbus Poll e obtidos pelo ESP32-01 via comunicação ESP-NOW, enviando-os para um computador de monitoramento remoto com PGCS embutido via mensagens NG. Além dos dados provenientes dos estados das saídas dos ativos, também são enviadas informações de identificação única, descrição técnica e funcionalidade dos mesmos, cujas características são essenciais nas camadas da arquitetura de referência para digitalizá-los.

Como diferencial, é possível mostrar a característica NG de arquitetura centrada a nomes, que permite ao usuário que as informações únicas dos ativos digitalizados sejam descritas com o uso de linguagem natural, garantindo uma maior aproximação e iteração homem-máquina, sendo um conceito fundamental para o futuro fabril denominado como Indústria 5.0. Por fim, os testes também tem o intuito de mostrar que o ICPS desenvolvido em uma arquitetura FI com uma estrutura baseada em nomeação tem como vantagem lidar com problemas relacionados às limitações da Internet atual, como desacoplamento ID/LOC, identificadores limitados pelo endereçamento IP e flexibilidade da rede.

Este Procedimento de Testes realiza uma análise da troca de mensagens NG entre o ESP32-02 e o computador, de modo a verificar os dados sendo recebidos pelo *host* para monitoramento. Sendo assim, a NG hospedada no computador de monitoramento e controle remoto possui a aplicação denominada *IoTTestApp* que é preparada para receber dados no par NG. Esses dados são armazenados em um arquivo de texto denominado *Data.txt* no diretório *novagenesis/IO/IoTTestApp*. Para a inicialização da aplicação no computador, o *script* responsável pela sua execução deve ser configurado com o endereço MAC do par que corresponde ao microcontrolador ESP32-02, de modo a prepará-lo para a comunicação NG. Esse script é denominado *frodo_iot_esp32_wifi.sh*, e foi aberto pelo editor de texto gedit para as configurações necessárias conforme mostrado na Figura 5.7.

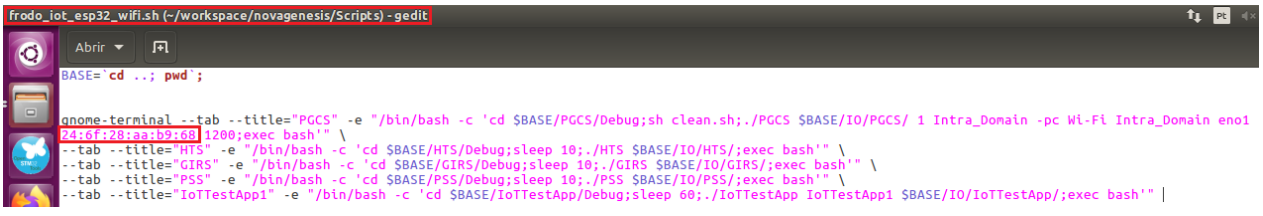

Figura 5.7: Configuração do endereço MAC do ESP32-02 na NG do computador de monitoramento remoto.

Como pode ser visto pela Figura 5.7, o endereço MAC do microcontrolador ESP32-02 foi configurado no *script*. Além disso, pode-se observar outras configurações, como por exemplo

a pilha de rede (Wi-Fi), o limitador de escopo das mensagens *Intra_Domain* e a interface de rede *eno1*.

Para o monitoramento da troca de mensagens, foi usado o *software* Wireshark, responsável por analisar o tráfego de rede, e organizá-lo por protocolos. A Seção 5.3.3 apresenta os Resultados do Procedimento de Testes da Parte 2.2 do Cenário Geral, a partir da inicialização do *script* e consequentemente da análise das mensagens NG.

5.2.4 Procedimento de Testes para Análise do Consumo de Memória e Energia dos Microcontroladores ESP32-01 e ESP32-02

Este Procedimento de Testes tem o intuito de avaliar o *hardware* usado no projeto, considerando o consumo de memória e de energia dos microcontroladores usados. O consumo de memória *Read-Only Memory* (ROM), em português, Memória Somente de Leitura, pode ser observado no processo de construção e inicialização do código, na própria interface das IDEs utilizadas para programar ambos os microcontroladores usados. Para isto, o objetivo é verificar a viabilidade de memória para alocar todo o programa nos *chips*, além da quantidade de espaço restante para possíveis alterações ou atualizações nos códigos.

Já para os testes relacionados ao consumo de memória *Random Access Memory* (RAM), em português, Memória de Acesso Aleatório, é importante notar que os 520 KB disponíveis pelo microcontrolador ESP32 são na verdade divididos em múltiplos tipos de RAM, como:

- Data Random Access Memory (DRAM), em português, Memória de Acesso Aleatório de Dados, usado para armazenar dados e acessado como *heap*;
- Instruction Random Access Memory (IRAM), em português, Memória de Acesso Aleatório de Instruções, usado somente para armazenar dados executáveis, como instruções específicas;
- Data/Instruction Random Access Memory (D/IRAM), em português, Memória de Acesso Aleatório de Dados/Instruções, que pode ser usado como qualquer um dos dois tipos de memória descritos anteriormente.

Desta forma, cada código aloca um espaço específico dos tipos de memória RAM no microcontrolador, variando de acordo com a aplicação, pois se o uso de IRAM aumentar muito, ele consome parte do DRAM disponível. Para esta análise, foi avaliada a memória *heap* referente à porção de memória alocada dinamicamente durante o programa, que é basicamente a forma de acesso à DRAM.

Por fim, para a análise energética do microcontrolador ESP32, a Figura 5.8 apresenta os principais módulos de consumo de energia do *chip* [123].

Figura 5.8: Principais módulos de consumo de energia do ESP32.

Baseado na Figura 5.8, é possível observar que o consumo de energia do chip baseia-se nos módulos: periféricos, core e memória do ESP32, Wi-Fi, Bluetooth (BT), rádio, co-processador *Ultra Low Power* (ULP), em português, consumo de energia ultra baixo e módulo *Real-Time Clock* (RTC), em português, Relógio de Tempo Real.

Para a análise de economia de energia, foram considerados 2 modos configuráveis existentes no ESP32: *Active Mode* e *Modem Sleep Mode*. O *active mode* consiste na ativação de todos os módulos do chip descritos anteriormente. Segundo o *datasheet* do ESP32, o consumo de energia neste modo pode variar sob as seguintes condições descritas na Tabela 5.1:

Tabela 5.1: Consumo de Energia do ESP32-02 no *Active Mode*.

Características usadas	**Consumo de Energia**
Pacote Tx Wi-fi (13dBm a 21dBm)	160mA a 260mA
Pacote Tx Wi-Fi/Bluetooth (0dBm)	120mA
Pacote Rx Wi-Fi/Bluetooth e Escuta	80mA a 90mA

A Figura 5.9 ilustra o resumo do *Active Mode* do ESP32, com módulos ativos e inativos e consumo de energia.

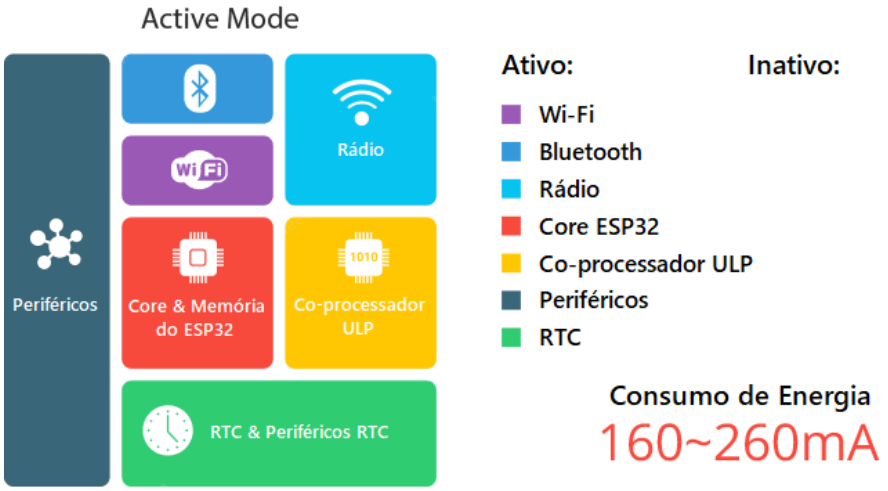

Figura 5.9: Modo Economia de Energia *ActiveMode* do ESP32.

No *Modem Sleep Mode*, tudo é ativo com excessão do Wi-Fi, Bluetooth, rádio Internet e periféricos. Para manter as conexões Wi-Fi e Bluetooth ativas, essas funções são ativadas em intervalos pré-definidos. Desta forma, este modo só é configurado quando o microcontrolador se conecta ao roteador no modo de estação, permanecendo conectado por meio do mecanismo de *beacon Delivery Traffic Indication Message* (DTIM), em português, Mensagem de indicação de tráfego de entrega.

Para economizar energia, o ESP32 desativa o módulo Wi-Fi entre dois intervalos de *beacon* DTIM e desperta automaticamente antes da próxima chegada de *beacon*, sendo o tempo de suspensão decidido pelo tempo de intervalo do *beacon* DTIM do roteador, sendo geralmente de 100ms a 1000ms. Assim sendo, neste modo o chip consome cerca de 3mA com o intervalo de tempo maior do módulo Wi-Fi desativado, e 20mA com o intervalo de tempo menor.

A Figura 5.10 mostra o resumo do *Modem Sleep Mode* do microcontrolador, com módulos ativos e inativos e consumo de energia.

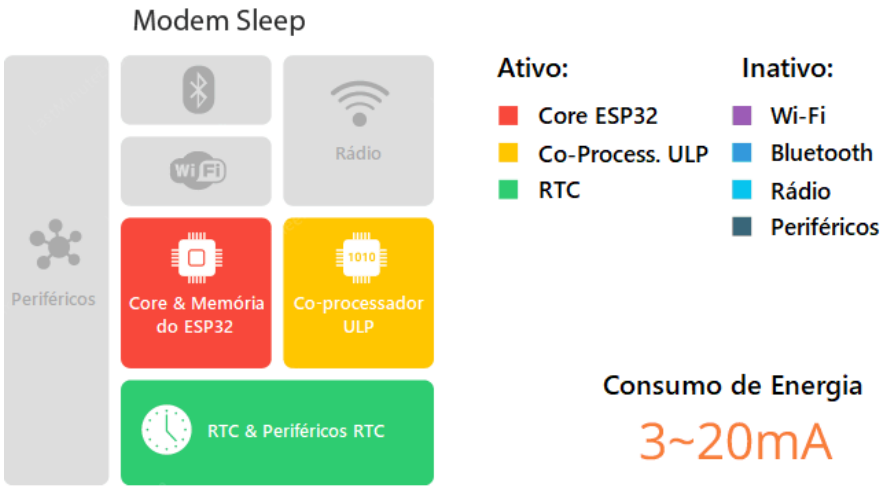

Figura 5.10: Modo Economia de Energia *ActiveMode* do ESP32.

Considerando todos esses dados, é importante destacar que, por padrão, o ESP32 possui como modo de economia de energia o *Modem Sleep Mode* configurado no mínimo, resultando no intervalo de tempo mínimo no qual o Wi-Fi é desativado. Com este modo padrão e o intervalo de *beacon* do roteador *Access Point* (AP), em português, Ponto de Acesso, configurado para 102,4ms, foi realizado um teste com período 2 horas para o ESP32-02 receber 1200 pacotes de dados via comunicação ESP-NOW do microcontrolador ESP32-01, sendo cada pacote transferido num intervalo de 6 segundos com os dados de saída dos ativos do ambiente industrial simulado a serem digitalizados. Desta forma, conforme visto no na Parte 2.1 do Cenário Geral, através do microcontrolador ESP32-01 é possível verificar a taxa de pacotes de dados entregues ao ESP32-02.

5.3 Resultados e Análises dos Procedimentos de Testes

Esta Seção apresenta as análises e resultados obtidos dos Procedimentos de Testes das Partes 1, 2.1 e 2.2 do Cenário Geral, além do Procedimento de Testes envolvendo o consumo de memória e energia dos microcontroladores.

5.3.1 Resultados do Procedimento de Testes da Parte 1 do Cenário Geral

Conforme detalhado no Procedimento de Testes da Parte 1 do Cenário Geral, para o teste de envio de dados ao *buffer*, as entradas %I0.0 e %I0.2 foram acionadas através de suas respectivas chaves. Como consequência, percebe-se que os LEDs sinalizadores do PLC correspondentes às entradas %I0.0 e %I0.2 e às saídas %Q0.0 e %Q0.2 foram acesos, indicando que elas foram acionadas. Consequentemente, a lâmpada conectada à saída %Q0.2 também

foi acesa. A Figura 5.11 mostra o teste para acionamento das saídas %Q0.0 e %Q0.2.

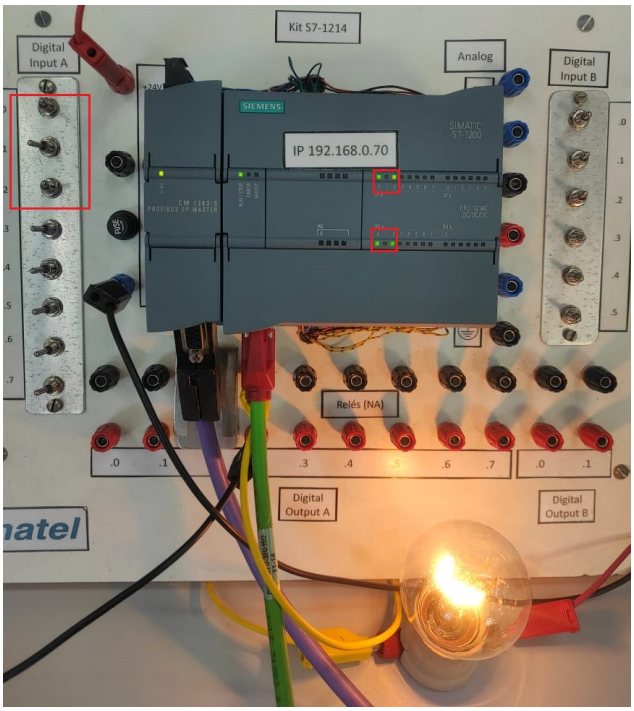

Figura 5.11: Acionamento das saídas %Q0.0 e %Q0.2 do PLC.

No programa do TIA Portal V13, é possível ver o resultado da ação. A Figura 5.12 ilustra as linhas de código das saídas %Q0.0 e %Q0.2 acionadas devido ao chaveamento das respectivas entradas.

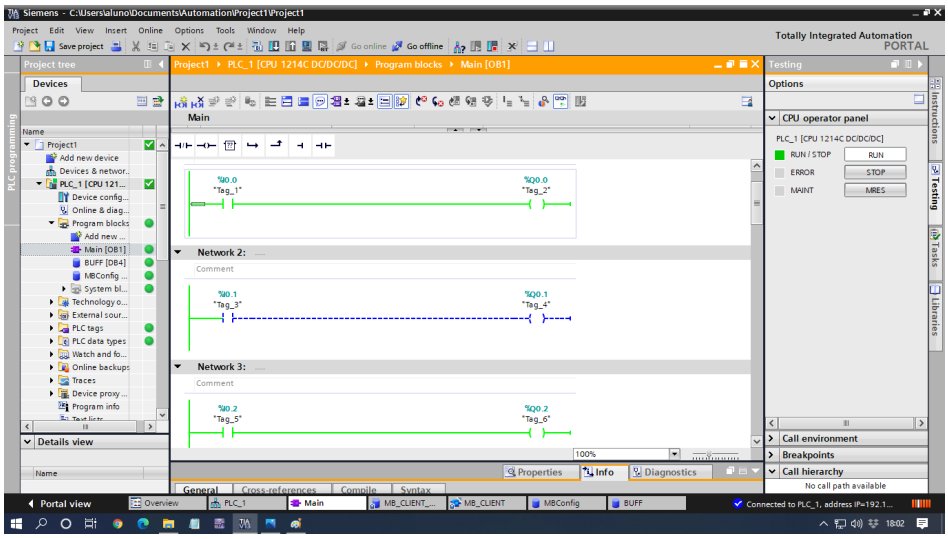

Figura 5.12: Acionamento das saídas %Q0.0 e %Q0.2 do PLC.

Como pode ser visto através da Figura 5.12, as linhas de %Q0.0 e %Q0.2 ficaram verdes devido ao acionamento das chaves conectadas às entradas digitais %I0.0 e %I0.2 resultando no acionamento das respectivas saídas digitais.

Além disso, os blocos *MOVE* cujos pinos habilitadores *EN* estão conectados aos contatos normalmente abertos de %Q0.0 e %Q0.2 também ficaram com as linhas verdes, sendo acionados devido ao nível lógico alto recebido nas saídas. A Figura 5.13 mostra o acionamento desses blocos. É importante ressaltar que, devido à saída %Q0.1 não ter sido acionada neste teste, seu respectivo bloco *MOVE* acionado em verde permanece sendo aquele cujo o habilitador *EN* está conectado a um contato normalmente fechado.

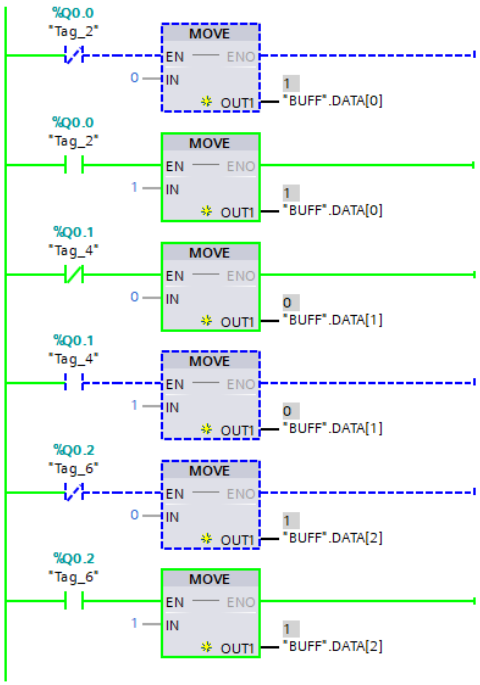

Figura 5.13: Acionamento dos Blocos *MOVE* com habilitadores %Q0.0 e %Q0.2 em contatos normalmente abertos.

Esses blocos são responsáveis por realizar o monitoramento do *buffer* de armazenamento dos estados das saídas digitais. Assim sendo, o valor inteiro "1" é movido para os endereços do *buffer* referentes às saídas digitais %Q0.0 e %Q0.2 que são respectivamente, *"BUFF".DATA[0]* e *"BUFF".DATA[2]*. Consequentemente, esses valores são direcionados ao pino *MB_DATA_PTR* do bloco *MB_Client* e transmitidos via protocolo Modbus TCP/IP. A Figura 5.14 ilustra o bloco de dados do *buffer* com os valores recebidos atualizados em tempo real.

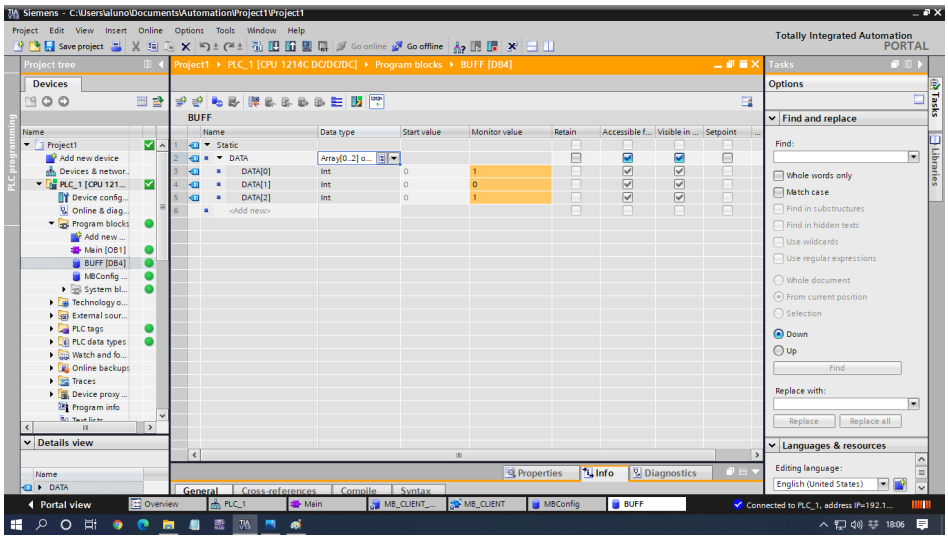

Figura 5.14: Bloco de Dados do Buffer Modbus TCP/IP com os valores recebidos atualizados.

Com base nos testes realizados na Parte 1 do Cenário Geral, é possível concluir que o código é flexível para que qualquer tipo de dado que possa ser transmitido via protocolo Modbus TCP/IP para posteriormente ser enviado via NG ao computador de monitoramento remoto. O programa permite que a aplicação possa inserir novas entradas e saídas digitais e analógicas para serem monitoradas simultaneamente, bastando apenas alterar a referência das variáveis no bloco de dados do *buffer*, sendo limitada apenas pelo *hardware*, devido à quantidade máxima de dispositivos que podem ser conectados a um módulo I/O do PLC. O programa também permite a flexibilidade no tipo de dado a ser transmitido, visto que o *buffer* pode conter diferentes tipos de dados, como por exemplo valores inteiros, words para valores analógicos, strings e até mesmo valores *booleanos* convertidos através do bloco *MOVE*. Assim sendo, este ambiente industrial real criado pode ser usado para testar a aquisição de dados dos ativos e o ICPS desenvolvido pela arquitetura NG, podendo até mesmo alterar a quantidade de dispositivos fabris para serem digitalizados pela aplicação.

Como já foi mencionado no Capítulo 4, devido aos equipamentos usados no ambiente industrial criado estarem disponíveis no Laboratório III-1 do Inatel e o computador para monitoramento remoto dos dados com o serviço PGCS da NG estar localizado no Laboratório ICT-Lab, foi adotado o uso do *software* Modbus Poll para simular o *buffer* de armazenamento do estados das saídas digitais do PLC como Modbus Cliente. O objetivo principal é realizar todos os processos em um mesmo ambiente físico e na mesma rede, afim de facilitar o gerenciamento e organização dos testes.

5.3.2 Resultados do Procedimento de Testes da Parte 2.1 do Cenário Geral

Os resultados do Procedimento de Testes da Parte 2.1 do Cenário geral consistem na aquisição de dados do acionamento das saídas %Q0.0 e %Q0.2 armazenadas no *buffer* do PLC, que foram simuladas no *software* Modbus Poll. Primeiramente, foi simulado o acionamento da saída %Q0.2 do PLC, que representava a lâmpada, em seguida o acionamento da saída %Q0.0, referente ao motor. A Figura 5.15 apresenta a configuração no *software* Modbus Poll para enviar o valor "1" ao Registrador Holding 2 referente à saída digital %Q0.2 (lâmpada).

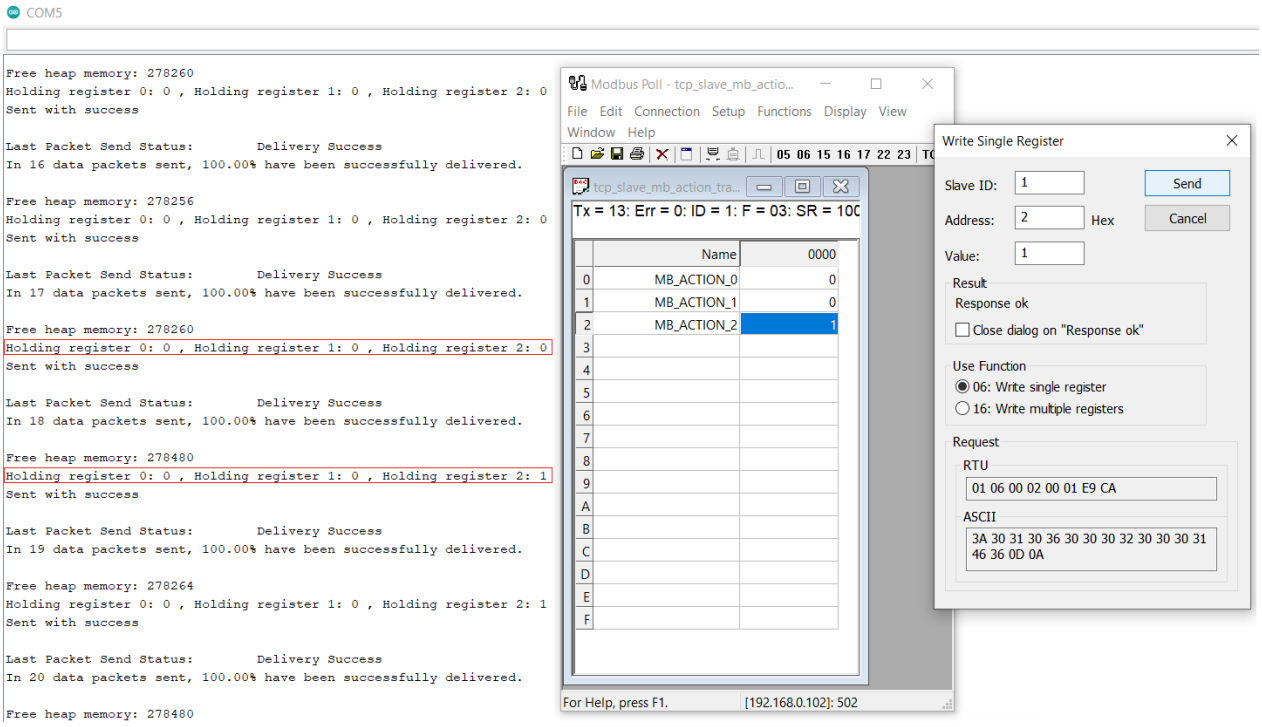

Figura 5.15: Simulação do acionamento da saída digital %Q0.2 do PLC (lâmpada).

Os Registradores Holding 0, 1 e 2 foram representados no *software* de simulação através das variáveis *MB_ACTION_0*, *MB_ACTION_1* e *MB_ACTION_2*. Como pode ser visto na Figura 5.15, o Registrador Holding de endereço 2 (*MB_ACTION_2*) referente à saída da lâmpada foi configurado com o valor "1". Com isso, o terminal monitor do ESP32-01 que estava recebendo os valores "0" nos 3 Registradores Holding passou a receber o valor "1" no endereço 2, conforme destacado na figura.

Finalmente, a etapa final consiste no acionamento da saída digital %Q0.0 (motor) que está sendo simulado através da variável *MB_ACTION_0* do *software* Modbus Poll. A Figura 5.16 mostra a configuração no simulador para enviar o valor "1" ao Registrador Holding 0 referente à saída digital %Q0.0 (motor).

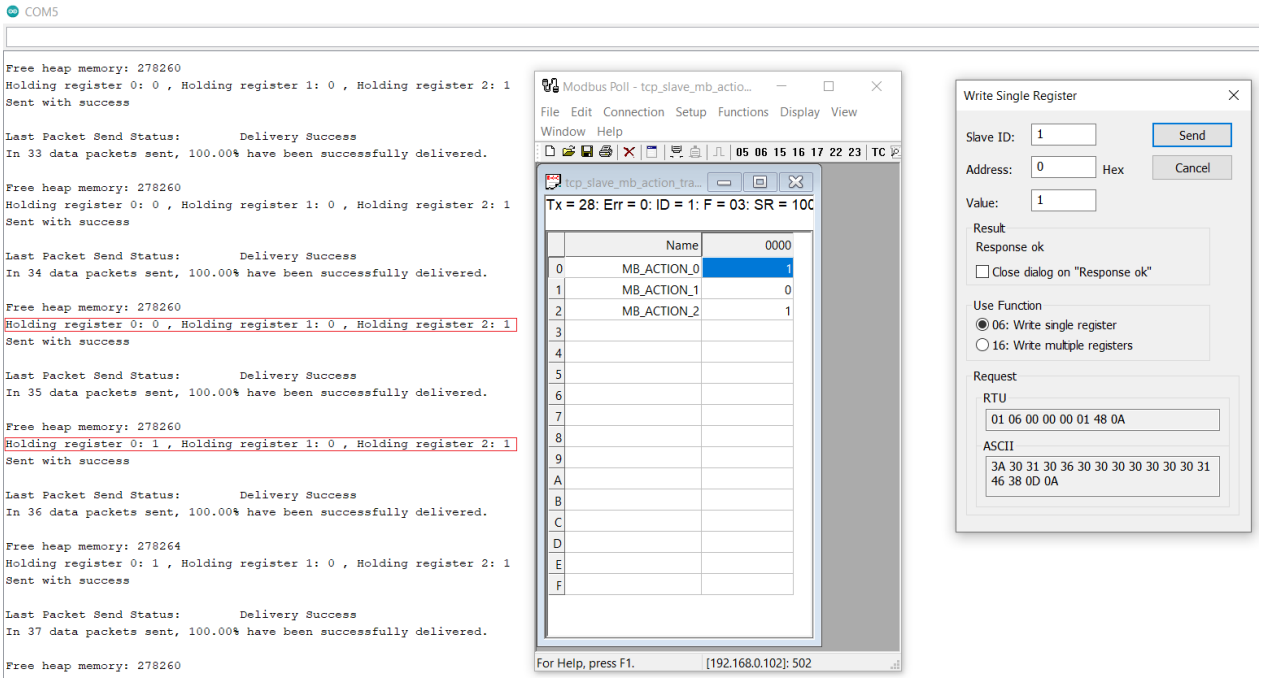

Figura 5.16: Simulação do acionamento da saída digital %Q0.0 do PLC (motor).

Conforme ilustrado na Figura 5.16, o Registrador Holding de endereço 0 (*MB_ACTION_0*) referente à saída do motor foi configurado com o valor "1". Desta forma, o terminal monitor do ESP32-01 que estava recebendo o valor "1" apenas no Registrador Holding 2 passou a receber o valor "1" também no Registrador Holding 0.

Baseado nesses resultados, é possível concluir que o uso compartilhado da comunicação ESP-NOW e Modbus TCP/IP permite a possibilidade de diferentes PLCs com diferentes módulos I/O comunicarem entre si através dos respectivos endereços MAC dos seus representantes microcontroladores ESP32, criando uma rede industrial com comunicação entre os dispositivos via comunicação ESP-NOW. Além disso, os resultados mostram a viabilidade de aquisição de dados de ativos industriais para serem armazenados e enviados ao ICPS desenvolvido pela NG, que será detalhado na próxima Seção.

Visto que a comunicação ESP-NOW se comunica apenas com microcontroladores ESP32, esta dissertação propõe a inclusão da arquitetura NG como ESP-NOW receptor de dados, ampliando o cenário de forma que qualquer equipamento com a NG embarcada possa se comunicar com os demais dispositivos, desde diferentes tipos de microcontroladores à máquinas *host*, como é o caso desta aplicação, que realiza o monitoramento remoto através de um computador com PGCS embutido. Desta forma, a NG entrega não somente os dados provenientes dos dispositivos industriais, mas também toda a estrutura de um ICPS baseado no modelo RAMI 4.0, com características inovadoras de auto-descoberta, oferta de serviços, contratação dinâmica e controle por *software* dentro de um contrato de serviço. Os resultados e vantagens referentes às novidades apresentadas pela NG como ICPS serão descritas com mais detalhes

na seção seguinte.

5.3.3 Resultados do Procedimento de Testes da Parte 2.2 do Cenário Geral

Após configurar o *script* conforme descrito na Seção 5.2.3, o mesmo foi executado para iniciar a comunicação NG. Para exibir somente as mensagens desejadas no Wireshark, o programa possui a opção de filtrar o tráfego pela interface de rede. Portanto, foi usado como filtro a interface (eno1) que corresponde à interface configurada no *script* para a comunicação NG. Além disso, o Wireshark foi filtrado para exibir apenas mensagem cujo campo Ethertype fosse igual a "0x1234", que é o número de protocolo atribuído por ora para os pacotes de mensagens NG. Após essas observações, foi iniciado o processo de captura dos pacotes.

Inicialmente, o computador de monitoramento remoto envia mensagens HELLO ao endereço de destino MAC configurado. A Figura 5.17 ilustra a mensagem de HELLO sendo enviada ao ESP32-02.

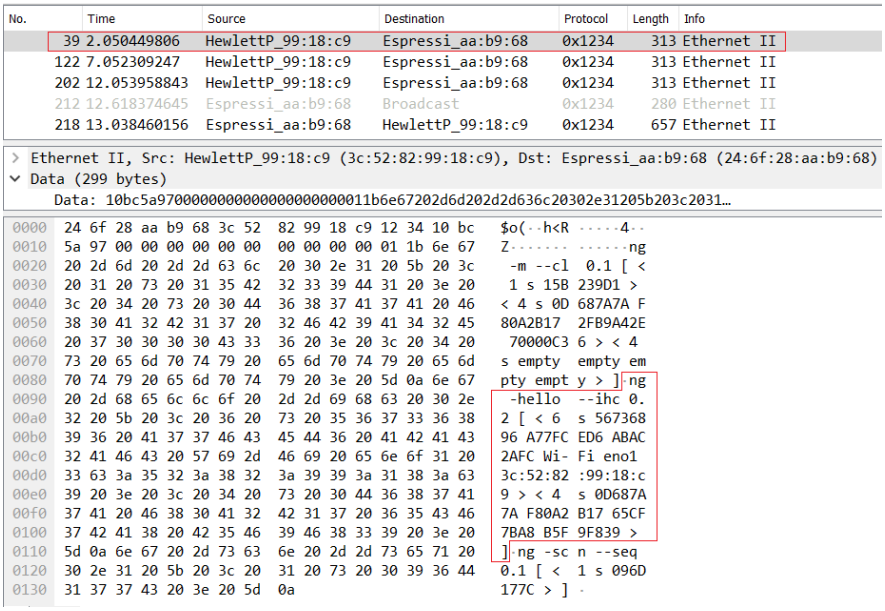

Figura 5.17: Mensagem de Hello sendo enviada ao ESP32-02.

Através das colunas de tempo (Time), endereço MAC de origem (Source) e endereço de Destino (Destination) da Figura 5.17, é possível identificar que os 3 primeiros pacotes capturados foram mensagens de Hello enviadas do computador ao ESP32-02, tendo aproximadamente 5 segundos de intervalo. Isso ocorre porque o PGCS foi configurado para emitir mensagens de Hello de 5 em 5 segundos para o endereço de destino. A linha de comando Hello foi destacada na mensagem, mostrando os argumentos e seus respectivos elementos. Quando o ESP32-02 finalmente é conectado, ele emite uma mensagem Broadcast de Hello a fim de indicar sua presença a outros dispositivos NG na rede. A Figura 5.18 mostra a

mensagem Broadcast de Hello do ESP32-02, destacando a linha de comando de Hello com suas informações inseridas.

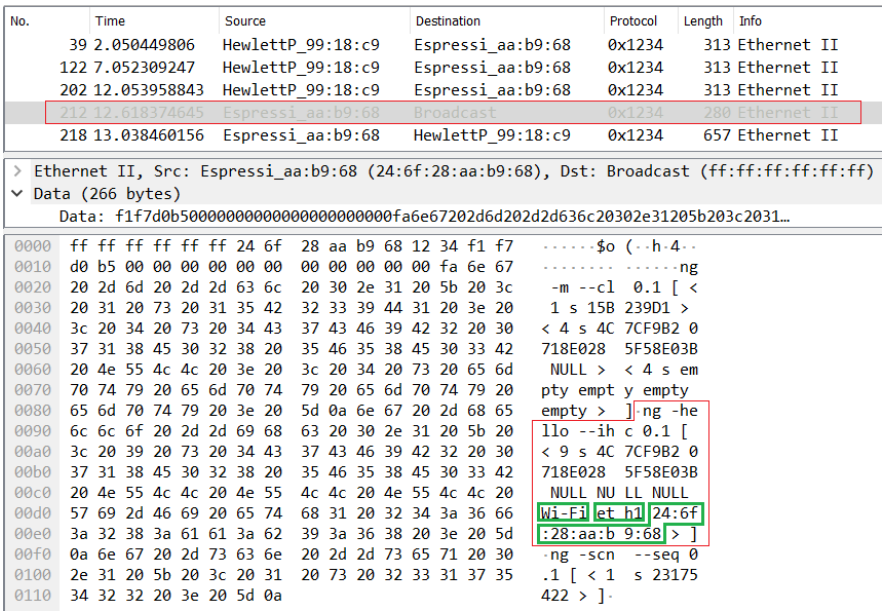

Figura 5.18: Mensagem Broadcast de Hello enviada pelo ESP32-02.

Como pode ser observado na Figura 5.18, as informações referentes à pilha e interface de rede, além do endereço MAC foram destacadas na mensagem. Estes dados permitem lidar com a flexibilidade da arquitetura e capacidade de lidar com questões envolvendo a heterogeneidade da rede, visto que os protocolos, pilhas e interface podem ser alterados para uma determinada aplicação e consequentemente suas informações de rede podem ser facilmente reprogramadas para serem expostas ao sistema. Após as trocas de mensagens de Hello, seguindo o ciclo de vida da NG, o ESP32-02 envia sua mensagem Exposition informando as palavras-chave do dispositivo. São enviados NBs entre os NLNs e SVNs das palavras-chave e os SVNs do Processo. A Figura 5.19 mostra o envio da mensagem Exposition, destacando as linhas de comando de Exposition com os NLNs das palavras-chave criadas no programa.

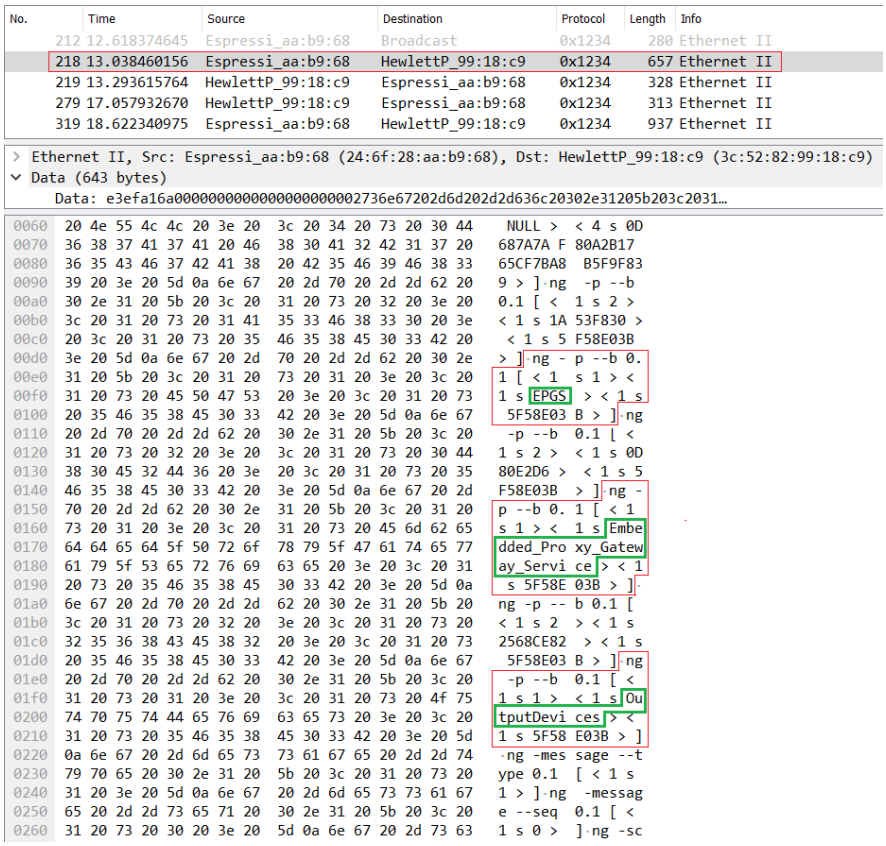

Figura 5.19: Mensagem Exposition enviada pelo ESP32-02.

Como pode ser observado pela Figura 5.19, as palavras-chave do dispositivo, destacadas em verde, foram enviadas ao computador de monitoramento remoto. Essas palavras-chave, juntamente com o endereço MAC, representam os identificadores escolhidos para o ativo a ser digitalizado, que é o PLC e suas saídas digitais monitoradas no seu *buffer* de armazenamento de estados. Percebe-se que a nomenclatura foi descrita em NLN, com o objetivo de se obter uma maior aproximação entre usuário e programa, que são conceitos primordiais em I5.0. Além disso, os identificadores são nomes únicos definidos independentemente da localização do mesmo, visto que seu localizador refere-se ao endereço IP. Desta forma, o sistema consegue lidar com as limitações da Internet atual de desacoplamento ID/LOC e de dispositivos em massa conectados em rede, visto que qualquer nome pode ser criado para identificá-lo, não se limitando ao enderaçamento IP.

Feito isso, o microcontrolador espera uma mensagem de confirmação (ack) de recebimento de Exposition pelo computador para enfim enviar sua mensagem SERVICE_OFFER. A Figura 5.20 mostra uma parte da mensagem, destacando as linhas de comando específicas da oferta de serviços.

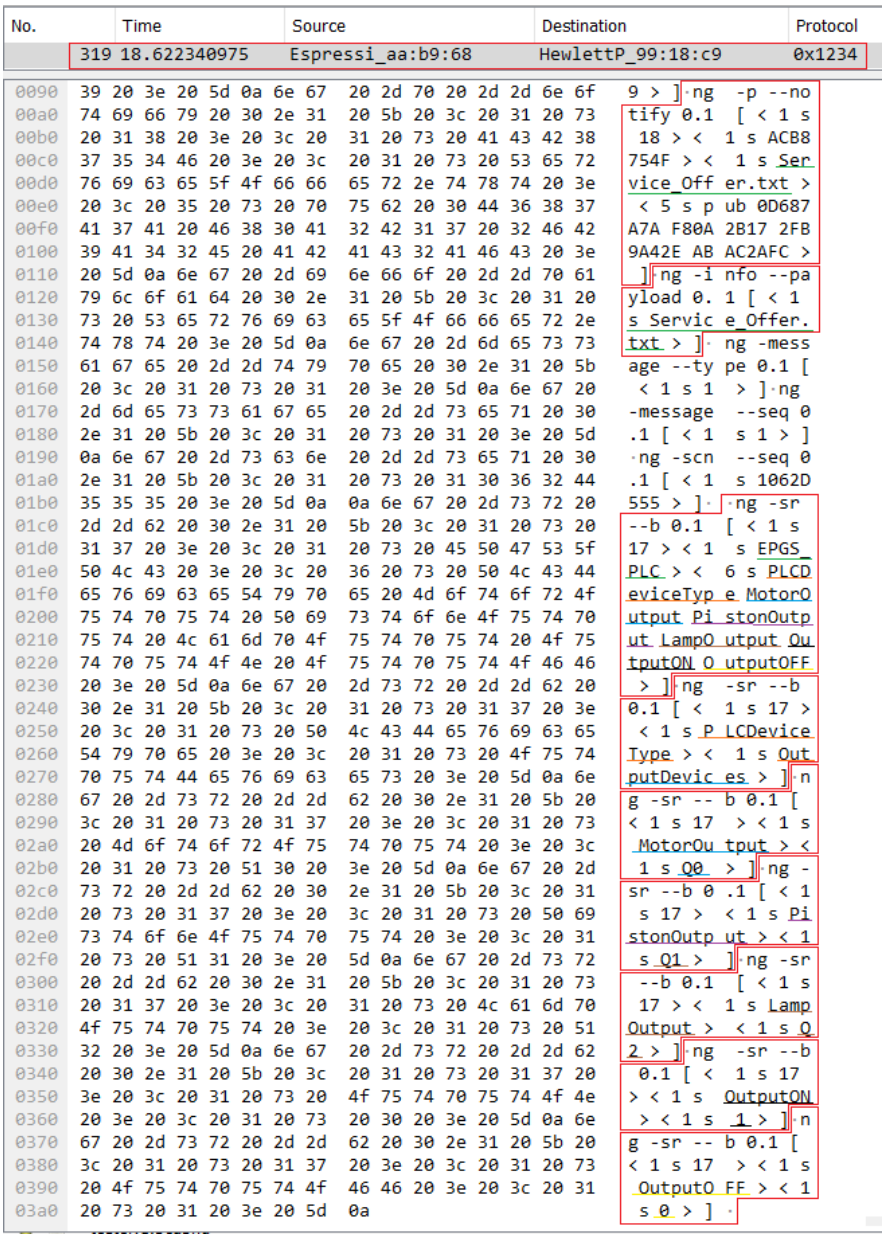

Figura 5.20: Mensagem de Oferta de Serviço enviada pelo ESP32-02.

Conforme destacado na Figura 5.20, NBs entre nomes e valores referentes às características e funcionalidades do dispositivo a ser monitorado são enviados. Essas características correspondem aos nomes e valores detalhados na Tabela 4.3 do Capítulo 4. Estas mensagens NG, por sua vez, são responsáveis por introduzir a descrição funcional e técnica dos ativos a serem digitalizados, sendo uma característica essencial do modelo de referência RAMI 4.0. Neste caso, foram expostos os tipo de dispositivos de saída e as possibilidades de valores que podem ser recebidos pelo mesmo, sendo 0 para *OFF* e 1 para *ON*. Quando o computador recebe a oferta de serviço, são enviados ao ESP32-02 uma confirmação do recebimento da mensagem

e uma mensagem de notificação para aceitação de serviço com uma chave de contrato. Ao receber essa mensagem, o microcontrolador envia uma mensagem para assinar a aceitação de serviço. A Figura 5.21 ilustra as etapas descritas anteriormente, destacando uma linha de comando de aceitação de serviço com o SVN da chave do contrato.

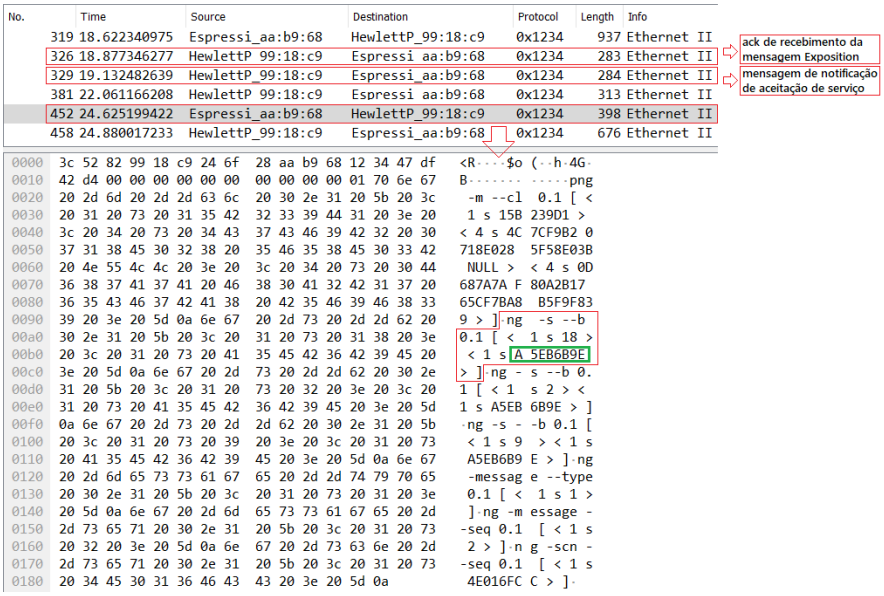

Figura 5.21: Mensagem de Recebimento de Notificação para Aceitação de Serviço e envio da Assinatura.

Após o envio da assinatura para aceitação do serviço, o ESP32-02 espera uma mensagem da entrega da aceitação de serviço para finalmente enviar suas publicações de dados. A Figura 5.22 mostra o envio da mensagem de publicação de dados do ESP32-02 ao computador de monitoramento remoto, destacando o nome do arquivo e os dados publicados equivalentes à simulação do *buffer* de armazenamento do PLC com dados provenientes dos estados das 3 saídas digitais.

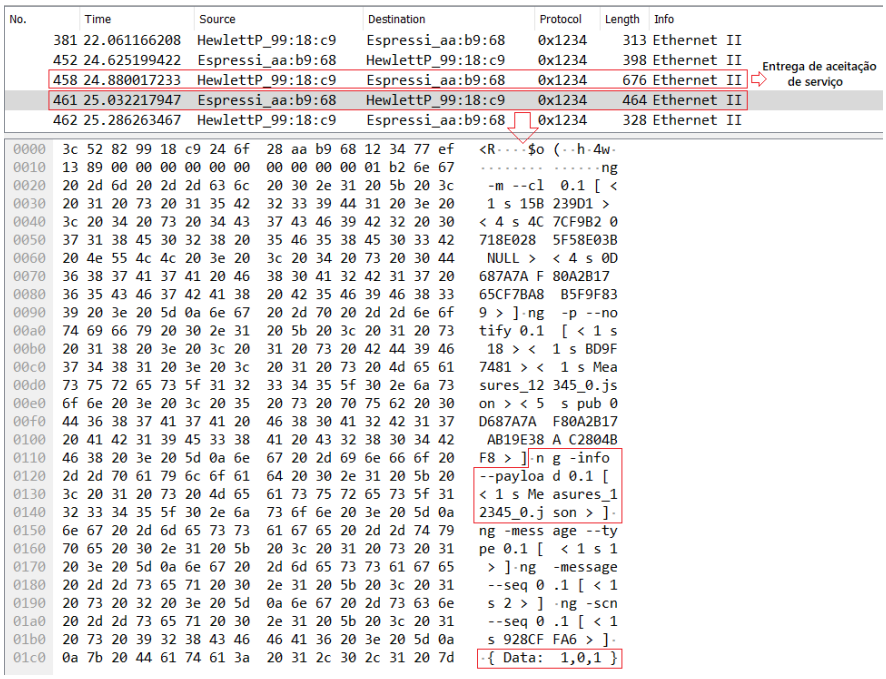

Figura 5.22: Mensagem de Publicação de dados.

Com base nos dados apresentados, pode-se observar pelas figuras que o intervalo de tempo entre as mensagens de origem do ESP32-02 são de aproximadamente 6 segundos. Esse valor se deve ao fato do intervalo de tempo de transmissão de pacotes de dados do ESP32-01 ao ESP32-02 ter sido configurado para 6 segundos. O motivo da adoção desse valor foi que, devido às mensagens NG possuírem um intervalo configurado médio de 5 segundos, esse tempo maior permite ao programa do microcontrolador ESP32-02 executar o estado atual da NG e então receber os dados via ESP-NOW para envio, já que tanto a arquitetura NG, quanto a comunicação ESP-NOW trabalham na camada física Wi-Fi. Desta forma, há uma sincronização, além de evitar possíveis falhas de transmissão de pacotes com *tasks* Wi-Fi simultâneas em execução.

Finalmente, a Figura 5.23 ilustra o resultado do monitoramento dos testes realizados, no qual foi simulado o *buffer* de armazenamento de estados com as 3 saídas digitais do PLC inicialmente desligadas, seguidos pelo acionamento da lâmpada (dado da Saída %Q0.2 do PLC enviado via Modbus TCP/IP pelo Registrador Holding 2) e finalmente do motor (dado da Saída %Q0.0 do PLC enviado via Modbus TCP/IP pelo Registrador Holding 0).

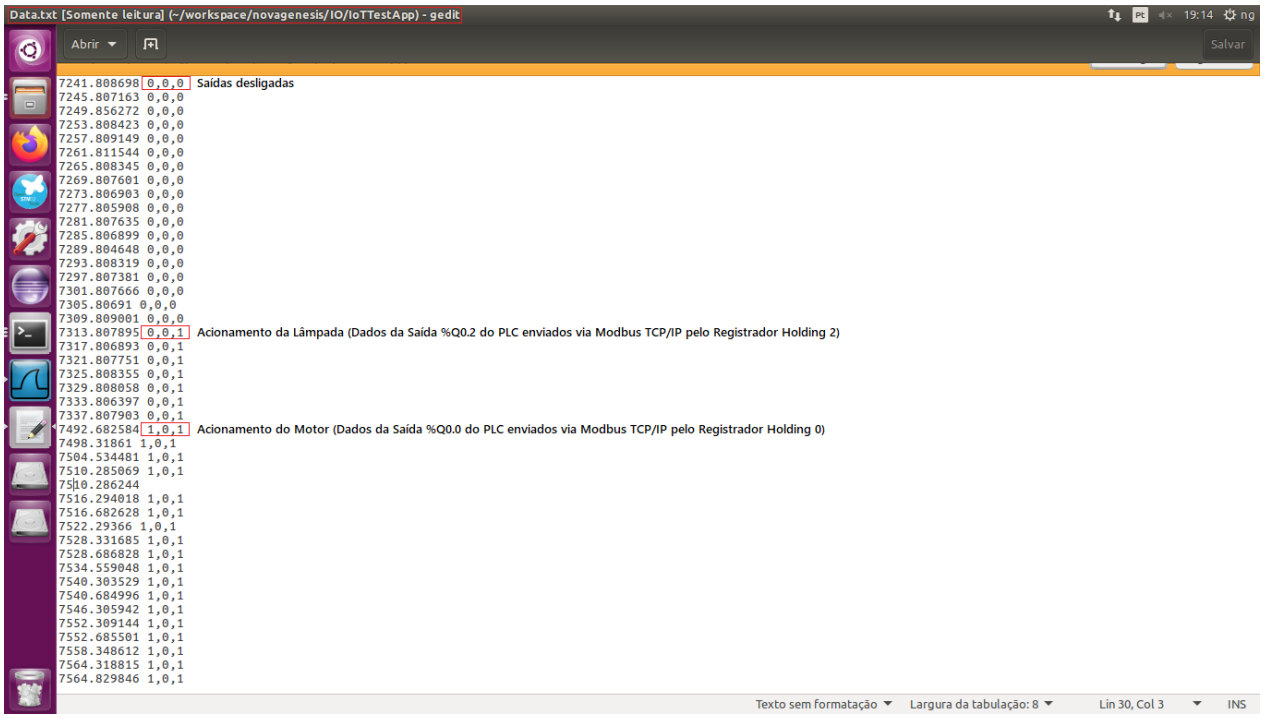

Figura 5.23: Mensagem de Publicação de dados.

Como pode ser observado pela Figura 5.23, os dados recebidos das mensagens NG através dos arquivos ".json" foram encaminhados para o arquivo "Data.txt", cujos valores são atualizados em tempo real. Desta forma, os resultados mostram que a aplicação para monitorar equipamentos simultaneamente pode ser realizada com a arquitetura NG.

Além disso, as demais características essenciais de um ICPS baseado no modelo RAMI 4.0 podem ser vistas através dos resultados, além da capacidade do sistema lidar com as questões de FI envolvendo as limitações da Internet atual. Conforme ilustrado na Figura 5.18 e comentado anteriormente, a NG mostra sua capacidade em lidar com a flexibilidade e heterogeneidade da rede com a facilidade de, através de sua arquitetura centrada a nomes, mostrar todas as informações implementadas no sistema ICPS, como pilha, protocolo e interface, podendo ser facilmente reprogramados com novas nomenclaturas para o caso de novas configurações da rede.

Já a Figura 5.19 propõe, além do endereço MAC, uma estrutura de nomes como identificadores únicos para o ativo a ser digitalizado, independente do seu localizador referente ao endereço IP, resolvendo problemas relacionados ao desacoplamento ID/LOC e consequentemente à limitação de endereçamento IPv4 para identificação de dispositivos na rede.

Como o foco deste trabalho é validar a NG como uma arquitetura FI capaz de, não somente lidar com as limitações da Internet atual discutidas ao longe desta dissertação, mas também implementar um ICPS baseado no RAMI 4.0, foi criada uma Seção à parte para realizar uma análise qualitativa dos benefícios da NG como proposta ao modelo RAMI 4.0 de

forma mais detalhada, comparando o sistema proposto com todas as camadas de arquitetura do modelo de referência. Esta análise será detalhada no Capítulo 6.

5.3.4 Resultados do Procedimento de Testes para Análise do Consumo de Memória e Energia do Microcontrolador ESP32

A seguir, serão apresentados os resultados dos testes referentes ao consumo de memória dos microcontroladores ESP32-01 e ESP32-02 usados neste projeto, seguidos pela análise de seu consumo de energia.

Análise do Consumo de Memória do Microcontrolador ESP32-01:

A Tabela 5.2 apresenta as informações relacionadas ao uso de memória do microcontrolador ESP32-01, considerando a Memória *flash* total 4 MB e a memória RAM de 520 KB:

Tabela 5.2: Uso de Memória do Microcontrolador ESP32-01.

Tipo de Memória	**Quantidade Disponível (Bytes)**
Memória Flash (ROM)	3,579815 MB
Memória RAM (heap)	282,168 KB

Através da Tabela 5.2, pode-se observar que a quantidade de memória flash usada para o código, correspondente à memória ROM, foi de 420,185 KB, ou seja, a disponibilidade de memória após a compressão do programa de aquisição de dados do ambiente industrial simulado foi de 3,579815 MB (89,4954%). Portanto, conclui-se que o código não ocupa muito espaço no dispositivo, podendo ser adicionadas novas funcionalidades e alterações no mesmo sem se preocupar com a limitação de memória *flash*. Em relação à memória *heap*, o espaço livre foi de 282,168 KB, sendo um valor baixo comparado com a memória RAM total do chip que é de 520 KB.

Análise do Consumo de Memória do Microcontrolador ESP32-02:

A Tabela 5.3 ilustra informações do uso de memória *flash* e RAM no microcontrolador ESP32-02.

Tabela 5.3: Uso de Memória do Microcontrolador ESP32-02.

Tipo de Memória	**Quantidade Disponível (Bytes)**
Memória Flash (ROM)	3,552 MB
Memória RAM (heap)	170,276 KB

Para a memória ROM, o código do programa ocupou um espaço de 448,439 KB, resultando em uma quantidade de 3,552 MB de memória flash livre (88,7890%). Pode-se concluir que o espaço ocupado pelo código é bem pequeno comparado com a memória flash total.

Já a memória *heap* foi analisada avaliando seu uso durante os diferentes estados NG para observar a porção de memória livre mínima que pode ser alocada dinamicamente durante a execução do programa. Para o cálculo da memória *heap* disponível, foi usada uma função para calcular seu valor mínimo. Desta forma, a função foi chamada ao fim de cada estado NG, desde "HELLO" à "PUB_DATA". Durante o término de cada estado, a função era chamada e a quantidade mínima de memória *heap* livre até aquele momento da execução do programa era atualizada. Ao fim de todos os estados NG, obteve-se o valor de 170,276 KB. Este valor é, na verdade, a quantidade mínima de memória *heap* livre durante toda a execução do programa, sendo usados cerca de 67,26% da memória RAM total no pior caso.

Análise dos Modos de Economia de Energia do ESP32:

A seguir, será apresentada uma análise do impacto dos modos de economia *Modem Sleep Mode* e *Active Mode* na taxa de pacotes de dados entregues ao microcontrolador ESP32-02. A Figura 5.24 mostra os resultados obtidos na taxa de pacotes de dados entregues ao ESP32-02 considerando o *Modem Sleep Mode*.

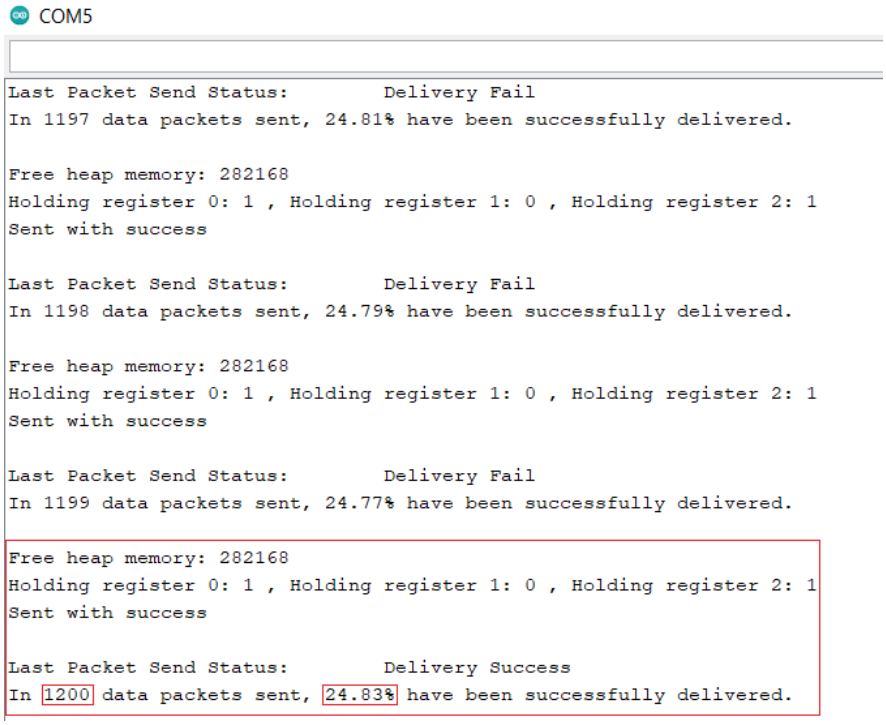

Figura 5.24: Taxa de Pacotes de dados entregues ao ESP32-02 considerando o *Modem Sleep Mode*.

Conforme mencionado na Seção 5.2.4 e visto na Figura 5.24, para os testes de consumo de energia foi feito um experimento de execução de toda a aplicação ICPS durante um período de 2 horas. Este teste realizou a transmissão de 1200 pacotes de dados independentes e simulados do *buffer* de armazenamento de estados das 3 saídas digitais monitoradas entregues ao microcontrolador ESP32-01, sendo posteriormente transmitidas ao ESP32-02, configurado em *Modem Sleep Mode*. Apesar do ESP32-01 enviar os pacotes de dados corretamente, apenas 24,83% das amostras coletadas foram entregues com sucesso. Isso se deve ao fato do modo de economia de energia ativado gerar uma perda de pacotes muito grande para esta aplicação.

Com os mesmos requisitos de testes realizados anteriormente, o microcontrolador ESP32-02 foi configurado para o *Active Mode*. A Figura 5.25 ilustra o terminal monitor da porta conectada ao ESP32-01 ao término das 1200 transmissões de pacotes de dados com o receptor no modo ativo.

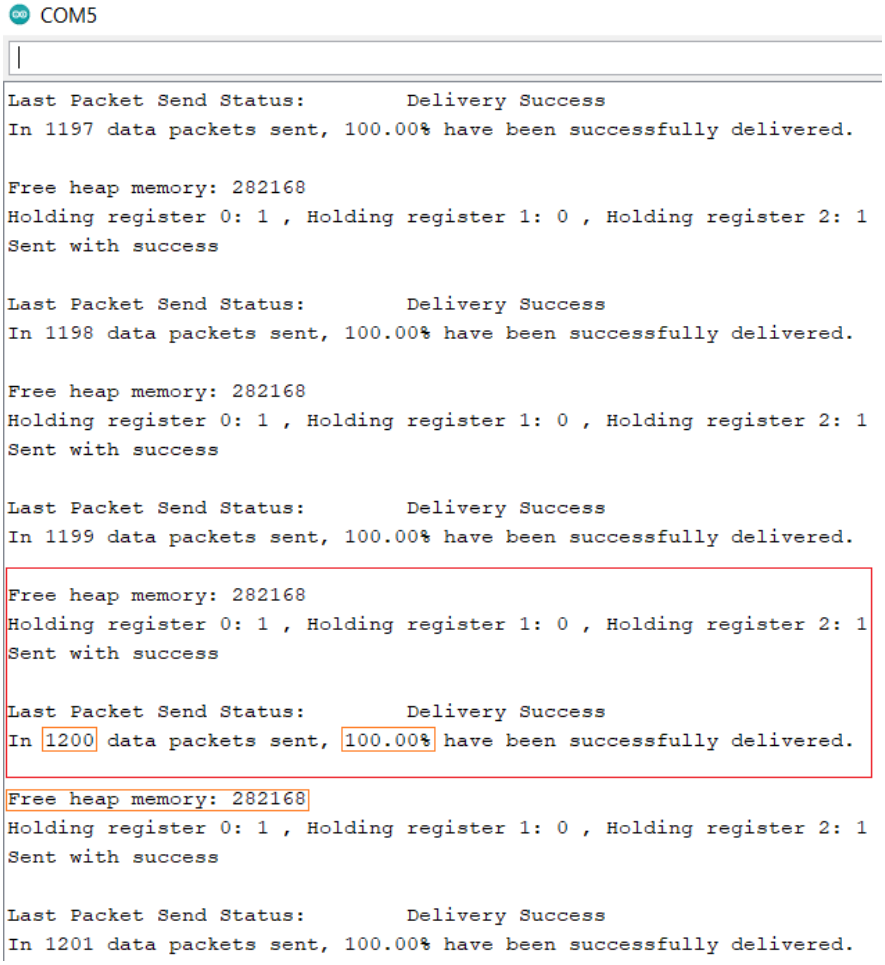

Figura 5.25: Transmissão de 1200 pacotes de dados recebidos pelo ESP32-01 via Modbus TCP/IP ao ESP32-02 com comunicação ESP-NOW.

Como pode ser observado pela Figura 5.25, dos 1200 pacotes de dados enviados, 100%

foram entregues com sucesso, mostrando o funcionamento do projeto na execução mútua dos protocolos usados com taxa máxima de pacotes de dados entregues neste modo.

Com isso, pode-se concluir que o uso do modo de economia *Modem Sleep Mode* resulta numa drástica redução do consumo de energia e consequentemente uma grande perda de pacotes. Essa configuração pode ser viável em aplicações cujos dispositivos monitorados apresentam uma alteração nos valores de suas entradas ou saídas em intervalos de tempo muito altos, e a taxa máxima de pacotes entregues com sucesso não é necessária. Neste caso, dado um grande período de tempo, basta apenas um pacote de dados entregue com sucesso para a atualização correta do valor de entrada ou saída do ativo, visto que o mesmo irá demorar para atualizar novamente.

Porém, para aplicações cujos dados dos dispositivos monitorados são atualizados em intervalos de tempo muito pequenos e a precisão da taxa de amostragem é um ponto importante a ser considerado, o ESP32-02 deve ser configurado no *Active Mode*. Um exemplo disso é, por exemplo, um sensor de temperatura, que dependendo do ambiente pode oscilar bastante em frações de segundos. Para este caso, uma alta taxa de perda de pacotes de dados reflete na perda de informações referentes ao monitoramento do ambiente naquele curto intervalo de tempo, podendo, dependendo da aplicação, ter sérias consequências para a indústria.

Capítulo 6

Análise Qualitativa dos Benefícios da NG como Proposta ao Modelo RAMI 4.0

6.1 Comparação do Cenário Geral com as Camadas do RAMI 4.0:

Visto que o objetivo principal desta dissertação é avaliar a NovaGenesis como uma arquitetura para o desenvolvimento de um modelo ICPS baseado no RAMI 4.0, este Capítulo tem como objetivo realizar uma análise qualitativa dos benefícios da NG como proposta à arquitetura de referência, e identificar os requisitos atendidos pelo sistema desenvolvido através da análise dos resultados obtidos pela aplicação. A Figura 6.1 ilustra o mapeamento do Cenário Geral implementado e suas respectivas contribuições como modelo para a arquitetura RAMI 4.0.

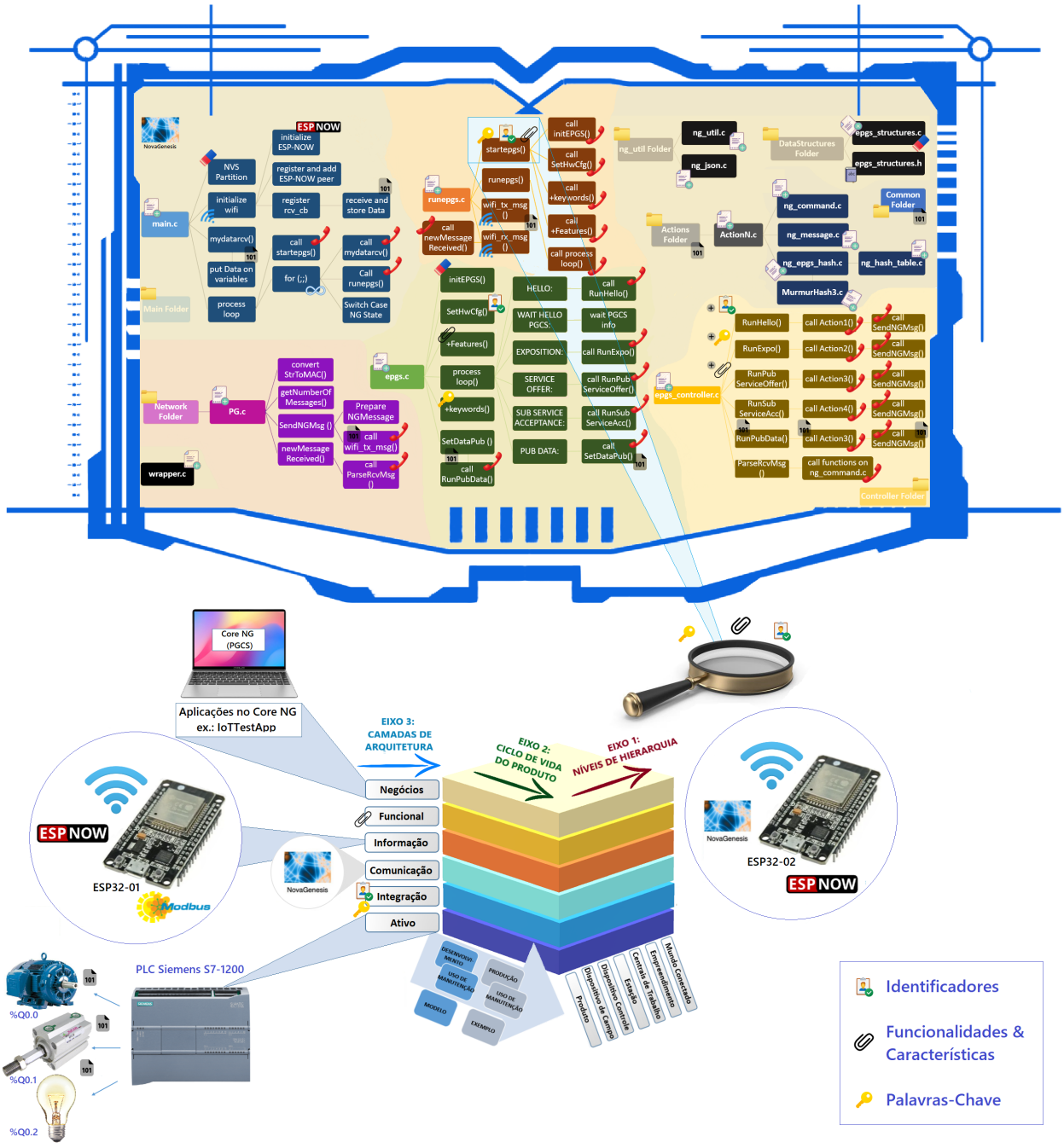

Figura 6.1: Cenário Geral comparado ao RAMI 4.0.

A Camada de Arquitetura Ativo, como o próprio nome sugere, é representada pelo próprio ativo físico, que no caso da aplicação corresponde ao PLC e seus respectivos dispositivos conectados ao módulo I/O (ex.: motor, lâmpada, pistão, etc.). Já a Camada de Integração, responsável pela integração do mundo físico ao virtual, pode ser atendida através da capa-

cidade da arquitetura NG embarcada ao ESP32-02 permitir a representação digital do ativo através de palavras-chaves e identificação única com informações sobre os IDs do hardware, processo, OS e do endereço MAC do dispositivo Essas informações são enviadas ao computador de monitoramento remoto através das mensagens de Hello e exposição de recursos.

A Camada de Comunicação, responsável pela comunicação padronizada de serviços e dados para a Camada de Informação, e de serviços e controle para a Camada de Integração, pode ser atendida através da arquitetura NovaGenesis.Isso é possível através do encaminhamento de pacotes de dados para os pares NG cujas aplicações solicitaram o serviço, através de mensagens NG encapsuladas e enviadas pela rede via Wi-fi, por exemplo. É importante destacar que, por se tratar de uma arquitetura FI moderna e flexível, com características como SDN, é possível programar o hardware para o encaminhamento de pacotes através de outras tecnologias (ex.: loRA, Ethernet, etc), tornando-se um modelo totalmente adaptável ao ambiente.

A Camada de Informação, cuja função é descrever serviços e dados que podem ser oferecidos de acordo com a funcionalidade técnica dos ativos, é alcançada através do *software* programado pelo PLC e pelo ESP32-01. Assim sendo, o PLC é capaz de filtrar a leitura de dados desejada no processo de produção através das informações inseridas no blocos de dados do buffer para o bloco *Modbus_Client*. Então, apenas os dados e serviços que podem ser oferecidos são enviados ao ESP32-01 via protocolo Modbus TCP, que os analisa para enviar ao ESP32-02 via comunicação ESP-NOW, trabalhando com um gateway entre o ativo (PLC e dispostivos I/O) e a arquitetura NG embarcada (ESP32-02). É importante notar que a comunicação ESP-NOW possibilita nesse caso a comunicação entre diferentes dispositivos I/O industriais de diferentes PLCs e equipamentos, desde que os mesmo possuam um microcontrolador ESP32 como representante, criando uma rede industrial interna. Já a NovaGenesis permite a expansão dessa rede a qualquer dispositivo NG embarcado, possibilitando o uso de diferentes microcontroladores e até mesmo hosts de monitoramento e controle, como o computador de monitoramento e controle remoto usado nesta aplicação.

A Camada Funcional, responsável por descrever as funcionalidades técnicas do ativo, é realizada através da oferta de serviços NG, que envia ao seu respectivo par todas as informações desejadas do equipamento. Essas informações, por sua vez, são recebidas no computador de monitoramento e controle remoto e encaminhadas à pasta HTS, que armazena os NBs e conteúdos. A Figura 6.2 ilustra os dados recebidos pelo computador de monitoramento e controle remoto armazenados na pasta HTS.

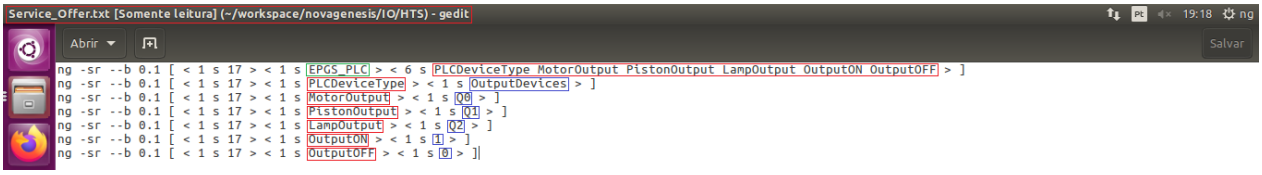

Figura 6.2: Funcionalidades técnicas do equipamento armazenadas no computador de monitoramento e controle remoto.

Como pode ser observado na Figura 6.2, são enviadas informações dos tipos de dispositivos sendo monitorados pelo buffer do PLC (dispositivos de saída, como motor, pistão e lâmpada), as relações de cada dispositivo com as saídas do módulo I/O (motor, pistão e lâmpada referentes, respectivamente, às saídas 0, 1 e 2) e as características das funcionalidades de monitoramento, que nesse caso trata-se de reconhecer o status da saída (acionado ou desacionado).

Por fim, a Camada de Negócios, responsável por organizar os serviços e criar processos de negócios e apoiar modelos de negócios sob restrições legais e regulatório pode ser atendida pelas aplicações NG. Desta forma, uma aplicação do par NG pode aceitar um serviço oferecido por outro dispositivo e então iniciar a comunicação e troca de dados através de um contrato, conforme foi descrito no ciclo de vida NG e seus estados. No caso deste projeto, a aplicação do computador de monitoramento e controle remoto responsável pela aceitação do serviço foi a *IoTTestApp*, que já é preparada para receber dados do ESP32. Assim sendo, a NovaGenesis permite a criação de diferentes aplicações para a comunicação entre os pares NG baseado no serviço desejado, admitindo inúmeras possibilidades e oportunidades relacionadas à trocas de dados.

A estrutura dessas camadas permite a criação do ponto-chave da arquitetura RAMI 4.0 denominado I4.0C, descrito na Seção 2.2, no qual os ativos são objetos globalmente e unicamente identificáveis com capacidade de compartilhamento de funcionalidades e descrição técnica, e comunicação para troca de dados.

Capítulo 7

Considerações Finais e Trabalhos Futuros

7.1 Considerações Finais

Este trabalho apresentou como proposta a aplicação de uma arquitetura de Internet do Futuro para Sistemas Ciber-Físicos na Indústria 4.0. Para isso, foram realizados testes em laboratório e simulações de um processo industrial legado com comunicação Modbus TCP/IP. O Cenário Geral da aplicação foi dividido em 2 partes, sendo a primeira parte responsável por criar um ambiente com dispositivos industriais reais e a segunda parte receber os dados desses ativos de forma simulada para realizar os testes do ICPS desenvolvido na arquitetura FI NG e baseado no modelo RAMI 4.0.

Primeiramente, o Capítulo 1 do trabalho teve como objetivo mostrar ao leitor a evolução industrial e o cenário atual, apontando como destaque a Indústria 4.0 e suas tecnologias, como IIoT e ICPS, e consequentemente a inclusão da Internet atual cada vez mais presente no ambiente fabril. Devido a este fator, foram mostradas as limitações da Internet atual e a necessidade do uso de arquiteturas emergentes capazes de suprir as exigências de I4.0, com suporte a tecnologias legadas e emergentes.

O Capítulo 2 apresentou o Background da dissertação, mostrando as informações essenciais para a compreensão desta dissertação. Nela, foram inseridos conceitos de Indústria 4.0 e suas principais tecnologias, como IIoT e CPSs, e apresentados os principais modelos de referência de arquiteturas, com destaque ao RAMI 4.0 que é focado ao setor de manufatura e digitalização fabril. Também foi detalhado o protocolo Modbus TCP/IP comumente usado em sistemas industriais legados e até mesmo emergentes. Por fim, foi discutido o tema de Internet do Futuro, cujas tecnologias podem ser úteis e eficientes no cenário industrial, e dada a presença em grande escala da Internet nos processos fabris, foi apresentada a ideia deste trabalho, que é a aplicação de uma arquitetura FI para ICPSs em I4.0 baseada no modelo RAMI 4.0. Como proposta, foi retratada a arquitetura NG.

O Capítulo 3 realizou um estado-da-arte dos projetos de ICPSs aplicados na indústria,

através de uma metodologia de seleção dos trabalhos que se identificam com tecnologias e propostas de I4.0 baseadas no modelo RAMI 4.0, como por exemplo representação virtual de ativos industriais e conexão de ativos à uma rede industrial com capacidade de armazenamento de dados em grande escala. Além da revisão literária, foi criada uma linha evolutiva dos trabalhos e uma relação com a arquitetura de referência. Por fim, foi analisada na revisão bibliográfica a ausência de propostas de ICPSs baseados no RAMI 4.0 que lidem com as limitações da Internet atual, dado o contexto da emersão de IIoT em I4.0. Portanto, foi discutida a proposta de uso de uma arquitetura FI (NG) para desenvolver um ICPS baseado na arquitetura de referência.

Para uma melhor organização e gerenciamento do projeto, a apresentação do Cenário Geral da tese foi dividida em 2 partes, que foram descritas detalhadamente no Capítulo 4 e analisadas através da realização de testes no Capítulo 5.

Na Parte 1, foi desenvolvido um processo industrial no laboratório III-1 do Inatel no qual chaves eletrônicas foram responsáveis pelo acionamento de 3 dispositivos de saídas digitais (lâmpada, pistão e motor). Os dispositivos I/O eram controlados pelo PLC Siemens S7-1200, sendo todos os equipamentos disponibilizados pelo instituto. Desta forma, foi programado um código no *software* TIA Portal capaz de realizar o monitoramento de um buffer que armazena o estado de 3 saídas digitais do PLC e tratar os dados para transmitir para a rede via protocolo Modbus TCP/IP. Observou-se que o programa permite a flexibilidade na leitura de equipamentos industriais, podendo monitorar entradas e saídas digitais e analógicas independente do tipo de variável (*booleana*, inteira ou real). Com isso, concluiu-se que este ambiente industrial real (físico) criado pode ser simulado como um ambiente de testes para avaliar o ICPS a ser desenvolvido na Parte 2 do Cenário Geral.

A Parte 2 do Cenário Geral consistia na inovação proposta pela dissertação através do desenvolvimento de um ICPS pela arquitetura NG baseado no modelo RAMI 4.0, e foi dividida em Partes 2.1 e 2.2. Para a Parte 2.1, foi usado o *software* Modbus Poll para simular o *buffer* de armazenamento dos estados das saídas digitais criadas na Parte 1 do Cenário Geral sem perda de representatividade. Com isso, foi concluído que o ambiente real físico pode ser substituído pelo seu respectivo simulador sem afetar os testes da hipótese em questão no trabalho. Então, foi utilizado um microcontrolador ESP32 (ESP32-01) para a aquisição dos dados provenientes do ambiente industrial simulado via comunicação Modbus TCP/IP e finalmente transmissão para o microcontrolador com a arquitetura NG embarcada (ESP32-02) via comunicação ESP-NOW. Os testes realizados mostraram a viabilidade de aquisição de dados do ambiente fabril e envio para o ICPS.

Para a Parte 2.2 do Cenário Geral, foi usado um microcontrolador ESP32 (ESP32-02) para receber os estados das saídas digitais e embarcar a arquitetura FI NG para desenvolver um ICPS baseado no modelo RAMI 4.0 capaz de lidar com as limitações da Internet atual. Os testes foram realizados pela análise das mensagens NG na rede através do *software* Wireshark e mostraram o envio dessas mensagens a um computador de monitoramento remoto com os dados referentes à digitalização dos ativos industriais, mostrando a viabilidade da NG na implementação do ICPS, atendendo aos requisitos do modelo de referência. Além disso, foram mostrados através dos testes a capacidade do Sistema Ciber-Físico Industrial

lidar com problemas da estrutura atual da Internet, como limitação de endereçamento de dispositivos em massa conectados, desacoplamento ID/LOC, heterogeneidade e flexibilidade da rede. Ainda, pôde-se observar novos benefícios e contribuições ao se usar a NG como um ICPS, como por exemplo a contratação dinâmica de programas, ciclo de vida dos serviços e representatividade do físico através de sua arquitetura centrada a nomes.

Ainda, foram analisados o consumo de memória e de energia do sistema. Em relação às informações de memória, foi concluído que o código de aquisição de dados do ambiente industrial programado no ESP32-01 consome uma porção de memória flash bem baixa, considerando 89,4954%, além de 282,168 KB de memória *heap* disponível. No caso do ESP32-02, também obteve-se um baixo uso de memória, com 88,7890% de memória flash disponível e uma quantidade mínima de memória *heap* livre durante toda a execução do programa de 170,276 KB.

Em relação ao consumo energético do sistema, foram realizados testes através da transmissão de 1200 pacotes de dados provenientes das saídas digitais simuladas, sendo enviados do ESP32-01 ao ESP32-02 durante um período de 2 horas. Primeiramente, o teste foi feito com o microcontrolador ESP32-02 no modo de economia *Modem Sleep Mode*, e a taxa de de pacotes de dados entregues com sucesso foi de 24,84%. Com isso, pôde-se concluir que o modo de economia deve ser usado apenas em aplicações cujo intervalo de tempo em que há mudanças e atualizações nos valores das saídas dos dispositivos a serem monitorados seja muito grande. Para o caso de exigência de altas taxas de amostragem, o *Active Mode* referente ao modo não-econômico deve ser usado, tendo como resultado uma taxa de pacotes entregues com sucesso de 100%.

Por fim, visto o objetivo principal desta dissertação de validar a viabilidade do uso da arquitetura FI NG para desenvolver um ICPS baseado no modelo RAMI 4.0, foi criado o Capítulo 6 destinado a realizar uma análise qualitativa dos benefícios da NG como proposta à arquitetura de referência. Pôde ser concluído que a NG atende aos requisitos das camadas de arquitetura do RAMI 4.0 sendo a aplicação uma proposta prática interessante ao modelo devido à sua flexibilidade programável e capacidade de digitalização de ativos industriais com a vantagem de lidar com as limitações da Internet atual devido à emersão de IIoT em I4.0. Além disso, a NG mostra o seu diferencial de, através de sua estrutura de nomeação, permitir ao usuário usar NLNs como identificadores da rede e dos ativos a serem digitalizados, além de sua descrição técnica e funcionalidades, aproximando o homem à máquina, sendo este conceito primordial para uma arquitetura inicial futura visando a Indústria 5.0.

7.2 Trabalhos Futuros

Vista a necessidade de se aplicar a arquitetura em ambientes mais robustos, é necessária a ampliação do cenário para monitorar não só diferentes dispositivos simultâneos em um módulo I/O específico, mas também um número maior de PLCs, de modo a analisar o comportamento da rede atuando com uma grande quantidade de dispositivos transmitindo e recebendo dados via protocolos Modbus TCP/IP e ESP-NOW. Assim sendo, tem-se como

trabalho futuro simular um aumento de sensores na rede e realizar análises estatísticas para mostrar o impacto desta escalabilidade. Estas análises podem ser feitas através de um tráfego de dados simulado (sintético) para mostrar que o sistema é escalável. Outro fator importante a ser considerado é o desenvolvimento da solução utilizando o ambiente real físico para testes e análises, de forma a avaliar o desempenho do trabalho sem o uso do simulador.

Além disso, também deve ser testada a implementação de uma rede industrial de dispositivos interna com diferentes PLCs comunicando entre si via representantes ESP32 através da comunicação ESP-NOW, desenvolvendo um sistema de filas para gerenciar os conteúdos de rede. A arquitetura NG também deve ser ampliada para que possa ser analisada a comunicação simultânea de diferentes dispositivos com NG embarcada e a eficiência com uma transmissão de pacotes de dados em grande escala.

Também é necessário criar uma solução mais versátil devido ao ESP-NOW se tratar de um protocolo comercial proprietário, e de haver protocolos diferentes do Modbus TCP sendo usados em ambientes industriais. Como a proposta desta dissertação se adequa ao Modbus TCP e ESP-NOW, há o desafio de elaborar uma solução flexível, já que a mesma teria que ser reestruturada para o uso de um protocolo diferente.

Por fim, tem-se como um desafio futuro o desenvolvimento para a execução do controle de dados, de modo que além da leitura, o core NG possa também realizar a escrita de dados através do envio de comandos de forma remota, resultando na atuação das saídas dos dispositivos industriais. Para isso, são necessárias alterações não somente no EPGS embarcado da NG disponível no microcontrolador ESP32-02, mas também no código do arquivo IoTTestApp do computador de monitoramento remoto para um novo modelo de aplicação capaz de enviar os comandos de controle. Além disso, além da estrutura de nomeação, pode-se trabalhar na NG como um modelo de arquitetura para atender aos requisitos futuros de Indústria 5.0, com aplicações que permitam uma maior iteração homem-máquina.

Referências Bibliográficas

[1] R. Kurzweil, *The Law of Accelerating Returns*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 381–416. [Online]. Available: https://doi.org/10.1007/978-3-662-05642-4_16

[2] G. Schuh, T. Potente, R. Varandani, C. Hausberg, and B. Fränken, "Collaboration moves productivity to the next level," *Procedia CIRP*, vol. 17, pp. 3 – 8, 2014, variety Management in Manufacturing. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2212827114003709

[3] Y. Liu, X. Ma, L. Shu, G. P. Hancke, and A. M. Abu-Mahfouz, "From industry 4.0 to agriculture 4.0: Current status, enabling technologies, and research challenges," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2020.

[4] T. D. Oesterreich and F. Teuteberg, "Understanding the implications of digitisation and automation in the context of industry 4.0: A triangulation approach and elements of a research agenda for the construction industry," *Computers in Industry*, vol. 83, pp. 121 – 139, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166361516301944

[5] E. Hofmann and M. Rüsch, "Industry 4.0 and the current status as well as future prospects on logistics," *Computers in Industry*, vol. 89, pp. 23 – 34, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166361517301902

[6] P. Dallasega, E. Rauch, and C. Linder, "Industry 4.0 as an enabler of proximity for construction supply chains: A systematic literature review," *Computers in Industry*, vol. 99, pp. 205 – 225, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166361517305043

[7] O. Cardin, "Classification of cyber-physical production systems applications: Proposition of an analysis framework," *Computers in Industry*, vol. 104, pp. 11 – 21, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166361517306231

[8] P. Hehenberger, B. Vogel-Heuser, D. Bradley, B. Eynard, T. Tomiyama, and S. Achiche, "Design, modelling, simulation and integration of cyber physical systems: Methods

and applications," *Computers in Industry*, vol. 82, pp. 273 – 289, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166361516300902

[9] R. F. Babiceanu and R. Seker, "Big data and virtualization for manufacturing cyber-physical systems: A survey of the current status and future outlook," *Computers in Industry*, vol. 81, pp. 128 – 137, 2016, emerging ICT concepts for smart, safe and sustainable industrial systems. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166361516300471

[10] H. Boyes, B. Hallaq, J. Cunningham, and T. Watson, "The industrial internet of things (iiot): An analysis framework," *Computers in Industry*, vol. 101, pp. 1 – 12, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166361517307285

[11] R. F. Babiceanu and R. Seker, "Cyber resilience protection for industrial internet of things: A software-defined networking approach," *Computers in Industry*, vol. 104, pp. 47 – 58, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166361517306954

[12] L. Wang, M. Törngren, and M. Onori, "Current status and advancement of cyber-physical systems in manufacturing," *Journal of Manufacturing Systems*, vol. 37, pp. 517 – 527, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0278612515000400

[13] GE, "Everything you need to know about the industrial internet of things," GE Digital, 2019, https://www.ge.com/digital/blog/everything-you-need-know-about-industrial-internet-things.

[14] C. Cassiolato, "Emi - interferência eletromagnética em instalações industriais e muito mais," Smar Technology Company, 2020, https://www.smar.com/brasil/artigo-tecnico/emi-interferencia-eletromagnetica-em-instalacoes-industriais-e-muito-mais.

[15] P. F. S. de Melo and E. P. Godoy, "Controller interface for industry 4.0 based on rami 4.0 and opc ua," in *2019 II Workshop on Metrology for Industry 4.0 and IoT (MetroInd4.0 IoT)*, 2019, pp. 229–234.

[16] D. Bogojevic and N. Gospic, "Some aspects of broadband internet growth and limits," in *2011 10th International Conference on Telecommunication in Modern Satellite Cable and Broadcasting Services (TELSIKS)*, vol. 2, 2011, pp. 615–618.

[17] S. Bradner, "The end of end-to-end security? [internet security]," *IEEE Security Privacy*, vol. 4, no. 2, pp. 76–79, 2006.

[18] L. Heuser and D. Woods, "Is europe leading the way to the future internet?" *IEEE Internet Computing*, vol. 14, no. 5, pp. 91–94, 2010.

[19] I. Lovrek, A. Caric, and D. Lucic, "Future network and future internet: A survey of regulatory perspective," in *2014 22nd International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 2014, pp. 186–191.

[20] A. M. Alberti, "Novagenesis: Convergent information architecture," Inatel, 2012, https://www.inatel.br/novagenesis/.

[21] X. T. Kong, R. Y. Zhong, Z. Zhao, S. Shao, M. Li, P. Lin, Y. Chen, W. Wu, L. Shen, Y. Yu, and G. Q. Huang, "Cyber physical ecommerce logistics system: An implementation case in hong kong," *Computers & Industrial Engineering*, vol. 139, p. 106170, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0360835219306394

[22] J. He, G. Jia, G. Han, H. Wang, and X. Yang, "Locality-aware replacement algorithm in flash memory to optimize cloud computing for smart factory of industry 4.0," *IEEE Access*, vol. 5, pp. 16 252–16 262, 2017.

[23] M. O. Gokalp, K. Kayabay, M. A. Akyol, P. E. Eren, and A. Koçyiğit, "Big data for industry 4.0: A conceptual framework," in *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2016, pp. 431–434.

[24] C. Shang, X. Bao, L. Fu, L. Xia, X. Xu, and C. Xu, "A novel key-value based real-time data management framework for ship integrated power cyber-physical system," in *2019 IEEE Innovative Smart Grid Technologies - Asia (ISGT Asia)*, 2019, pp. 854–858.

[25] E. Manavalan and K. Jayakrishna, "A review of internet of things (iot) embedded sustainable supply chain for industry 4.0 requirements," *Computers & Industrial Engineering*, vol. 127, pp. 925 – 953, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0360835218305709

[26] A. Karmakar, N. Dey, T. Baral, M. Chowdhury, and M. Rehan, "Industrial internet of things: A review," in *2019 International Conference on Opto-Electronics and Applied Optics (Optronix)*, 2019, pp. 1–6.

[27] W. Matsuda, M. Fujimoto, T. Aoyama, and T. Mitsunaga, "Cyber security risk assessment on industry 4.0 using ics testbed with ai and cloud," in *2019 IEEE Conference on Application, Information and Network Security (AINS)*, 2019, pp. 54–59.

[28] M. A. Kaleem and M. Khan, "Significance of additive manufacturingfor industry 4.0with introduction of artificial intelligence in additive manufacturing regimes," in *2020 17th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, 2020, pp. 152–156.

[29] J. B. Han, K. M. Yang, D. H. Kim, and K. H. Seo, "A modeling and simulation based on the multibody dynamics for an autonomous agricultural robot," in *2019 7th International Conference on Control, Mechatronics and Automation (ICCMA)*, 2019, pp. 137–143.

[30] G. D'Emilia, A. Di Ilio, A. Gaspari, E. Natale, R. Perilli, and A. G. Stamopoulos, "The role of measurement and simulation in additive manufacturing within the frame of industry 4.0," in *2019 II Workshop on Metrology for Industry 4.0 and IoT (MetroInd4.0 IoT)*, 2019, pp. 382–387.

[31] M. Lorenz, S. Knopp, and P. Klimant, "Industrial augmented reality: Requirements for an augmented reality maintenance worker support system," in *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, 2018, pp. 151–153.

[32] T. Lins and R. A. R. Oliveira, "Cyber-physical production systems retrofitting in context of industry 4.0," *Computers & Industrial Engineering*, vol. 139, p. 106193, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S036083521930662X

[33] C. e. S. Ministério da Indústria, "Indústria 4.0," Ministério da Indústria, Comércio e Serviços, 2019, http://www.industria40.gov.br/.

[34] P. I. 4.0, "What is indústria 4.0?" Federal Ministry of Education and Research, 2019, https://www.plattform-i40.de/PI40/Navigation/EN/Industrie40/WhatIsIndustrie40/what-is-industrie40.html.

[35] G. Campeanu, "A mapping study on microservice architectures of internet of things and cloud computing solutions," in *2018 7th Mediterranean Conference on Embedded Computing (MECO)*, 2018, pp. 1–4.

[36] A. Gilchrist, *Industry 4.0: The Industrial Internet of Things*. Apress Media, 2016, ch. 1, p. 1.

[37] GE, "Everything you need to know about the industrial internet of things," GE Digital, 2019, https://www.ge.com/digital/blog/everything-you-need-know-about-industrial-internet-things.

[38] N. Chaabouni, M. Mosbah, A. Zemmari, and C. Sauvignac, "A onem2m intrusion detection and prevention system based on edge machine learning," in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–7.

[39] A. Gilchrist, *Industry 4.0: The Industrial Internet of Things*. Apress Media, 2016, ch. 1, pp. 3–4.

[40] M. Venturelli, "A internet das coisas na indústria 4.0," Automação Industrial, 2017, https://www.automacaoindustrial.info/internet-das-coisas-na-industria-4-0/.

[41] C. Assawaarayakul, W. Srisawat, S. D. N. Ayuthaya, and S. Wattanasirichaigoon, "Integrate digital twin to exist production system for industry 4.0," in *2019 4th Technology Innovation Management and Engineering Science International Conference (TIMES-iCON)*, 2019, pp. 1–5.

[42] S. Akhtari, F. Pickhardt, D. Pau, A. D. Pietro, and G. Tomarchio, "Intelligent embedded load detection at the edge on industry 4.0 powertrains applications," in *2019 IEEE 5th International forum on Research and Technology for Society and Industry (RTSI)*, 2019, pp. 427–430.

[43] P. Poór, J. Basl, and D. Zenisek, "Predictive maintenance 4.0 as next evolution step in industrial maintenance development," in *2019 International Research Conference on Smart Computing and Systems Engineering (SCSE)*, 2019, pp. 245–253.

[44] C. Occhiuzzi, S. Amendola, S. Nappi, N. D'Uva, and G. Marrocco, "Rfid technology for industry 4.0: Architectures and challenges," in *2019 IEEE International Conference on RFID Technology and Applications (RFID-TA)*, 2019, pp. 181–186.

[45] HPE, "O que é iiot?" Hewlett Packard Enterprise, 2019, https://www.hpe.com/br/pt/what-is/industrial-iot.html.

[46] E. Lee, "Cyber-physical systems - are computing foundations adequate?" 01 2006.

[47] Intel, "Aproveitando o poder da iot (internet das coisas)," Intel - Look Inside, 2019, https://www.intel.com.br/content/dam/www/public/lar/br/pt/documents/articles/harnessing-the-power-of-iot-por.pdf.

[48] N. A. of Science and a. Engineering, *Cyber-Physical Systems: Driving force for innovation in mobility, health, energy and production*, 01 2011.

[49] E. A. Lee, "Cyber physical systems: Design challenges," in *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, May 2008, pp. 363–369.

[50] H. Boyes, B. Hallaq, J. Cunningham, and T. Watson, "The industrial internet of things (iiot): An analysis framework," *Computers in Industry*, vol. 101, pp. 1 – 12, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166361517307285

[51] B. Bagheri, S. Yang, H.-A. Kao, and J. Lee, "Cyber-physical systems architecture for self-aware machines in industry 4.0 environment," *IFAC-PapersOnLine*, vol. 48, no. 3, pp. 1622 – 1627, 2015, 15th IFAC Symposium onInformation Control Problems inManufacturing. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2405896315005571

[52] G. Cheng, L. Liu, X. Qiang, and Y. Liu, "Industry 4.0 development and application of intelligent manufacturing," in *2016 International Conference on Information System and Artificial Intelligence (ISAI)*, June 2016, pp. 407–410.

[53] R. Alguliyev, Y. Imamverdiyev, and L. Sukhostat, "Cyber-physical systems and their security issues," *Computers in Industry*, vol. 100, pp. 212 – 223, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166361517304244

[54] P. Fantini, G. Tavola, M. Taisch, J. Barbosa, P. Leitao, Y. Liu, M. S. Sayed, and N. Lohse, "Exploring the integration of the human as a flexibility factor in cps enabled manufacturing environments: Methodology and results," in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, 2016, pp. 5711–5716.

[55] Z. Liu, Z. Wang, Y. Ren, Q. Feng, D. Fan, and Z. Zuo, "A city medical resources distribution optimization platform based on cyber physical systems (cps)," in *2018 International Conference on Sensing,Diagnostics, Prognostics, and Control (SDPC)*, 2018, pp. 269–273.

[56] G. Kale and A. Patil, "Prototype architecture to control building energy usage using plc and cps," in *2018 International Conference on Current Trends towards Converging Technologies (ICCTCT)*, 2018, pp. 1–3.

[57] C. Shih, J. Chou, N. Reijers, and T. Kuo, "Designing cps/iot applications for smart buildings and cities," *IET Cyber-Physical Systems: Theory Applications*, vol. 1, no. 1, pp. 3–12, 2016.

[58] D. P. F. Möller and H. Vakilzadian, "Cyber-physical systems in smart transportation," in *2016 IEEE International Conference on Electro Information Technology (EIT)*, 2016, pp. 0776–0781.

[59] K. Antonopoulos, C. Panagiotou, C. P. Antonopoulos, and N. S. Voros, "A-farm precision farming cps platform," in *2019 10th International Conference on Information, Intelligence, Systems and Applications (IISA)*, 2019, pp. 1–3.

[60] M. U. Khan, S. Li, Q. Wang, and Z. Shao, "Cps oriented control design for networked surveillance robots with multiple physical constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 5, pp. 778–791, 2016.

[61] A. Zanni, "Sistemas cyber-físicos e cidades inteligentes," IBM Developer, 2015, https://www.ibm.com/developerworks/br/library/ba-cyber-physical-systems-and-smart-cities-iot/index.html.

[62] M. Hermann, T. Pentek, and B. Otto, "Design principles for industrie 4.0 scenarios: A literature review," 01 2015.

[63] J. Li, F. R. Yu, G. Deng, C. Luo, Z. Ming, and Q. Yan, "Industrial internet: A survey on the enabling technologies, applications, and challenges," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1504–1526, 2017.

[64] P. Monteiro, M. Carvalho, F. Morais, M. Melo, R. J. Machado, and F. Pereira, "Adoption of architecture reference models for industrial information management systems," in *2018 International Conference on Intelligent Systems (IS)*, Sep. 2018, pp. 763–770.

[65] K. Schweichhart, "Reference architectural model industrie 4.0 (rami 4.0) - an introduction," Platform Industrie 4.0, 2016, https://ec.europa.eu/futurium/en/system/files/ged/a2-schweichhart-reference_architectural_model_industrie_4.0_rami_4.0.pdf.

[66] P. I. 4.0, "Rami 4.0 – a reference framework for digitalisation," Platform Industrie 4.0, 2018, https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/rami40-an-introduction.pdf?__blob=publicationFile&v=3.

[67] S.-W. Lin, B. Murphy, E. Clauer, U. Loewen, R. Neubert, G. Bachmann, M. Pai, and M. Hankel, "Architecture alignment and interoperability: An industrial internet consortium and plattform industrie 4.0 joint whitepaper," Industrial Internet Consortium, Tech. Rep., 2017, https://www.iiconsortium.org/pdf/JTG2_Whitepaper_final_20171205.pdf.

[68] L. Silva, P. Pedreiras, P. Fonseca, and L. Almeida, "On the adequacy of sdn and tsn for industry 4.0," in *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*, 2019, pp. 43–51.

[69] J. Ordonez-Lucena, J. F. Chavarria, L. M. Contreras, and A. Pastor, "The use of 5g non-public networks to support industry 4.0 scenarios," in *2019 IEEE Conference on Standards for Communications and Networking (CSCN)*, 2019, pp. 1–7.

[70] B. e.V, "Implementation strategy industrie 4.0: Report on the results of the industrie 4.0 platform," Bitkom, 2015, https://www.zvei.org/fileadmin/user_upload/Presse_und_Medien/Publikationen/2016/januar/Implementation_Strategy_Industrie_4.0_-_Report_on_the_results_of_Industrie_4.0_Platform/Implementation-Strategy-Industrie-40-ENG.pdf.

[71] IBM, "Mqtt: The standard for iot messaging," GE Digital, 2020, https://mqtt.org.

[72] A. Amâncio, "O protocolo coap para iot," Wire Engenharia, 2018, http://wireengenharia.com.br/br/o-protocolo-coap-para-iot/.

[73] I. Acromag, "Introduction to modbus tcp/ip," Technical Reference – Modbus TCP/IP, 2005, https://www.prosoft-technology.com/kb/assets/intro_modbustcp.pdf.

[74] N. I. CORP, "O protocolo modbus em detalhes," Tech. Rep., 2021, https://www.ni.com/pt-br/innovations/white-papers/14/the-modbus-protocol-in-depth.html.

[75] J. Berg, "Introduction to modbus tcp/ip," Modbus Details, 2019, https://minimalmodbus.readthedocs.io/en/stable/modbusdetails.html.

[76] H. Jung and S. Koh, "Problem statements and requirements for mobile oriented future internet," in *ICTC 2011*, 2011, pp. 610–611.

[77] A. M. Alberti, G. D. Scarpioni, V. J. Magalhães, A. Cerqueira S., J. J. P. C. Rodrigues, and R. da Rosa Righi, "Advancing novagenesis architecture towards future internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 215–229, 2019.

[78] R. Atkinson, S. Bhatti, and S. Hailes, "Evolving the internet architecture through naming," *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 8, pp. 1319–1325, 2010.

[79] H. Lee, Y. Han, and S. Min, "Node mobility support scheme between the mobile network and pmipv6 networks," in *2009 First International Conference on Networks Communications*, 2009, pp. 93–97.

[80] R. Grandl, D. Han, S.-B. Lee, H. Lim, M. Machado, M. Mukerjee, and D. Naylor, "Supporting network evolution and incremental deployment with xia," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, p. 281–282, Aug. 2012. [Online]. Available: https://doi.org/10.1145/2377677.2377731

[81] S. YiLiang and Y. Zhenghong, "Communication rules learning strategy in big data network based on svn neural network," in *2016 International Conference on Intelligent Transportation, Big Data Smart City (ICITBS)*, 2016, pp. 309–313.

[82] A. Alberti, M. Casaroli, D. Singh, and R. Righi, "Naming and name resolution in the future internet: Introducing the novagenesis approach," *Future Generation Computer Systems*, vol. 67, 09 2016.

[83] K. K. Ramakrishnan, "Software-based networks: Leveraging high-performance nfv platforms to meet future communication challenges," in *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, 2016, pp. 24–24.

[84] D. Gu, X. Liu, G. Qin, S. Yan, Z. Luo, and B. Yan, "Vnet6: Soa based on ipv6 virtual network," in *2012 International Conference on Cloud and Service Computing*, 2012, pp. 32–39.

[85] S. Arshad, M. A. Azam, M. H. Rehmani, and J. Loo, "Recent advances in information-centric networking-based internet of things (icn-iot)," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2128–2158, 2019.

[86] M. Abbasi, H. Rezaei, V. G. Menon, L. Qi, and M. R. Khosravi, "Enhancing the performance of flow classification in sdn-based intelligent vehicular networks," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–10, 2020.

[87] M. Gasparyan, T. Braun, and E. Schiller, "L-scn: Layered scn architecture with supernodes and bloom filters," in *2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*, 2017, pp. 899–904.

[88] M. J. F. Alenazi, A. Almutari, S. Almowuena, W. Abdul, and E. K. Çetinkaya, "Nfv provisioning in large-scale distributed networks with minimum delay," *IEEE Access*, pp. 1–1, 2020.

[89] M. Baugher, B. Davie, A. Narayanan, and D. Oran, "Self-verifying names for read-only named data," in *2012 Proceedings IEEE INFOCOM Workshops*, 2012, pp. 274–279.

[90] A. M. Alberti and D. Singh, "Developing a novagenesis architecture model for service oriented future internet and iot: An advanced transportation system scenario," in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, 2014, pp. 359–364.

[91] X. Duan, Z. Yan, G. Geng, and B. Yan, "Dnsledger: Decentralized and distributed name resolution for ubiquitous iot," in *2018 IEEE International Conference on Consumer Electronics (ICCE)*, 2018, pp. 1–3.

[92] A. M. Alberti, "Novagenesis: Visão geral e detalhes de implementação," in *Conference: EduTecCria 2017*, Santa Rita do Sapucaí, Brazil, 2017, pp. 1–29.

[93] D. Wang, D. Chen, B. Song, N. Guizani, X. Yu, and X. Du, "From iot to 5g i-iot: The next generation iot-based intelligent algorithms and 5g technologies," *IEEE Communications Magazine*, vol. 56, no. 10, pp. 114–120, 2018.

[94] D. Miller, "Blockchain and the internet of things in the industrial sector," *IT Professional*, vol. 20, no. 3, pp. 15–18, 2018.

[95] F. A. B. Juárez, "Cybersecurity in an industrial internet of things environment (iiot) challenges for standards systems and evaluation models," in *2019 8th International Conference On Software Process Improvement (CIMPS)*, 2019, pp. 1–6.

[96] Nak-Jung Choi, Ji-In Kim, and Seok-Joo Koh, "Domain-based identifier-locator mapping management for distributed mobility control," in *2017 International Conference on Information Networking (ICOIN)*, 2017, pp. 674–678.

[97] Nakjung Choi, T. You, J. Park, Taekyoung Kwon, and Y. Choi, "Id/loc separation network architecture for mobility support in future internet," in *2009 11th International Conference on Advanced Communication Technology*, vol. 01, 2009, pp. 78–82.

[98] Z. Zhang, J. Lu, L. Xia, S. Wang, H. Zhang, and R. Zhao, "Digital twin system design for dual-manipulator cooperation unit," in *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, vol. 1, 2020, pp. 1431–1434.

[99] F. Biesinger and M. Weyrich, "The facets of digital twins in production and the automotive industry," in *2019 23rd International Conference on Mechatronics Technology (ICMT)*, 2019, pp. 1–6.

[100] J. Lee, M. Azamfar, J. Singh, and S. Siahpour, "Integration of digital twin and deep learning in cyber-physical systems: towards smart manufacturing," *IET Collaborative Intelligent Manufacturing*, vol. 2, no. 1, pp. 34–36, 2020.

[101] J. Sengupta, S. Ruj, and S. Das Bit, "A secure fog based architecture for industrial internet of things and industry 4.0," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2020.

[102] B. Kaynak, S. Kaynak, and O. Uygun, "Cloud manufacturing architecture based on public blockchain technology," *IEEE Access*, vol. 8, pp. 2163–2177, 2020.

[103] P. Fraga-Lamas, P. Lopez-Iturri, M. Celaya-Echarri, O. Blanco-Novoa, L. Azpilicueta, J. Varela-Barbeito, F. Falcone, and T. M. Fernández-Caramés, "Design and empirical validation of a bluetooth 5 fog computing based industrial cps architecture for intelligent industry 4.0 shipyard workshops," *IEEE Access*, vol. 8, pp. 45 496–45 511, 2020.

[104] W. Liu and J. Su, "A solution of dynamic manufacturing resource aggregation in cps," in *2011 6th IEEE Joint International Information Technology and Artificial Intelligence Conference*, vol. 2, Aug 2011, pp. 65–71.

[105] B. KARAKOSTAS, "A dns architecture for internet of things: A case study in transport logistics," *Procedia Computer Science*, vol. v. 19, pp. p. 594–601, 2013.

[106] N. Jazdi, "Cyber physical systems in the context of industry 4.0," 05 2014, pp. 1–4.

[107] I. Ungurean, N. Gaitan, and V. G. Gaitan, "An iot architecture for things from industrial environment," in *2014 10th International Conference on Communications (COMM)*, May 2014, pp. 1–4.

[108] M. Yu, M. Zhu, G. Chen, J. Li, and Z. Zhou, "A cyber-physical architecture for industry 4.0-based power equipments detection system," in *2016 International Conference on Condition Monitoring and Diagnosis (CMD)*, Sep. 2016, pp. 782–785.

[109] I. Grangel-González, L. Halilaj, G. Coskun, S. Auer, D. Collarana, and M. Hoffmeister, "Towards a semantic administrative shell for industry 4.0 components," in *2016 IEEE Tenth International Conference on Semantic Computing (ICSC)*, Feb 2016, pp. 230–237.

[110] C2NET, "Cloud collaborative manufacturing networks," H2020-FoF-1-2014: Process Optimisation of Manufacturing Assets, 2016, http://c2net-project.eu/documents/10184/12410/C2NET+Project_v3.pdf/138aea4c-d4c0-4591-b22c-d47fe99e64d7.

[111] M. Pisching, A. Tasca, M. Pessoa, F. Junqueira, and P. Miyagi, "Arquitetura para desenvolvimento de sistemas ciber-físicos aplicados na indústria 4.0," 10 2017.

[112] J. Jiang, "An improved cyber-physical systems architecture for industry 4.0 smart factories," in *2017 International Conference on Applied System Innovation (ICASI)*, May 2017, pp. 918–920.

[113] S. Yun, J. Park, and W. Kim, "Data-centric middleware based digital twin platform for dependable cyber-physical systems," in *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2017, pp. 922–926.

[114] I. Ayatollahi, J. Brier, B. Mörzinger, M. Heger, and F. Bleicher, "Soa on smart manufacturing utilities for identification, data access and control," *Procedia CIRP*, vol. 67, pp. 162 – 166, 2018, 11th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 19-21 July 2017, Gulf of Naples, Italy. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S221282711731137X

[115] A. Paulo and S. Oliveira, "Nodei4.0:integrando sistemas legados à indústria 4.0," *Revista de Engenharia e Pesquisa Aplicada*, vol. 2, 12 2017.

[116] X. F. Liu, M. R. Shahriar, S. N. A. Sunny, M. C. Leu, and L. Hu, "Cyber-physical manufacturing cloud: Architecture, virtualization, communication, and testbed," *Journal of Manufacturing Systems*, vol. 43, pp. 352 – 364, 2017, high Performance Computing and Data Analytics for Cyber Manufacturing. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0278612517300432

[117] CREMA, "Cloud-based rapid elastic manufacturing," FoF-01-2014 - Process optimisation of manufacturing assets, 2017, https://www.crema-project.eu/project/index.html.

[118] U. Kannengiesser and H. Müller, "Multi-level, viewpoint-oriented engineering of cyber-physical production systems: An approach based on industry 4.0, system architecture and semantic web standards," in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Aug 2018, pp. 331–334.

[119] M. R. Nascimento Marques Junior, B. Konzgen Maciel, G. Balota, R. Fonseca, M. Simosa, H. Conceição, E. M. Gonçalves, and S. Botelho, "Embedded agent based on cyber physical systems: Architecture, hardware definition and application in industry 4.0 context," 07 2018, pp. 584–591.

[120] M. Pisching, "Arquitetura para descoberta de equipamentos de manufatura com foco na indústria 4.0," 2018.

[121] V. J. Magalhães, "Contribuições no desenvolvimento do serviço novagenesis embarcado para internet das coisas," Master's thesis, Instituto Nacional de Telecomunicações, 2017.

[122] G. D. Scarpioni, "Integração do serviço novagenesis embarcado em um sistema operacional orientado a eventos em hardware real," Master's thesis, Instituto Nacional de Telecomunicações, 2017.

[123] L. M. Engineers, "Insight into esp32 sleep modes their power consumption," ESP32 Sleep Modes, 2021, https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/.

Apêndice A

Código do Microcontrolador ESP32-01

Este apêndice contém as linhas de código do microcrontrolador ESP32-01 para configuração como dispositivo Servidor Modbus para receber os dados das saídas digitais do PLC via comunicação Modbus TCP/IP, e configuração como ESP-NOW transmissor para enviar os dados armazenados ao ESP32-02 via protocolo ESP-NOW. O código foi programado em linguagem C++ no software Arduino IDE 1.8.13.

O apêndice foi citado na Seção 4.8.

A.1 Código de programação do arquivo ESP3202.ino

```
/*
  Configuration of an ESP32 microcontroller (ESP32-01) as Modbus Server for receiving data from devices
  via Modbus TCP/IP communication, and as ESP-NOW Sender for sending the stored
  data to another ESP32 (ESP32-02) via ESP-NOW protocol.

  Author: Diego Gabriel Soares Pivoto
*/

#include <WiFi.h>
#include <ModbusIP_ESP8266.h>
#include <esp_now.h>

uint8_t MACAddr[] = {0x24, 0x6F, 0x28, 0xAA, 0xB9, 0x68}; //ESP32-02 MAC Address

String success;

```

```
17 //Structure example to send data
18 //Must match the receiver structure
19 typedef struct struct_message {
20     int HR_0;
21     int HR_1;
22     int HR_2;
23 } struct_message;
24
25 // Create a struct_message called HR_Writings to hold Holding Registers
   ↪  and send them.
26 struct_message HR_Writings;
27
28 // Callback when data is sent
29 void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
30   Serial.print("\r\nLast Packet Send Status:\t");
31   Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" :
   ↪  "Delivery Fail");
32   if (status ==0){
33     success = "Delivery Success :)";
34   }
35   else{
36     success = "Delivery Fail :(";
37   }
38 }
39
40 // Modbus Registers Offsets
41
42 const int reg0_HREG = 0;
43 const int reg1_HREG = 1;
44 const int reg2_HREG = 2;
45
46
47 //ModbusIP object
48 ModbusIP mb;
49
50 void setup() {
51   Serial.begin(115200);
52
53   // Set device as a Wi-Fi Station
54   WiFi.mode(WIFI_STA);
55
56   // Init ESP-NOW
57   if (esp_now_init() != ESP_OK) {
```

```
58      Serial.println("Error initializing ESP-NOW");
59      return;
60    }
61
62    // Once ESPNow is successfully Init, we will register for Send CB to
63    // get the status of Trasnmitted packet
64    esp_now_register_send_cb(OnDataSent);
65
66    // Register peer
67    esp_now_peer_info_t peerInfo;
68    memcpy(peerInfo.peer_addr, MACAddr, 6);
69    peerInfo.channel = 1;
70    peerInfo.encrypt = false;
71
72    // Add peer
73    if (esp_now_add_peer(&peerInfo) != ESP_OK){
74      Serial.println("Failed to add peer");
75      return;
76    }
77    WiFi.begin("OpenWrt", "ng#gandalf*2016");
78
79    while (WiFi.status() != WL_CONNECTED) {
80      delay(500);
81      Serial.print(".");
82    }
83
84    Serial.println("");
85    Serial.println("WiFi connected");
86    Serial.println("IP address: ");
87    Serial.println(WiFi.localIP());
88
89    mb.server();
90
91    mb.addHreg(reg0_HREG, 0);
92    mb.addHreg(reg1_HREG, 0);
93    mb.addHreg(reg2_HREG, 0);
94  }
95
96  void loop() {
97     //Call once inside loop() - all magic here
98     mb.task();
99     Serial.print("Holding register 0: ");
100    Serial.print(mb.Hreg(reg0_HREG));
```

```
101     Serial.print(" , Holding register 1: ");
102     Serial.print(mb.Hreg(reg1_HREG));
103     Serial.print(" , Holding register 2: ");
104     Serial.println(mb.Hreg(reg2_HREG));
105
106   // Set values to send
107   HR_Writings.HR_0 = mb.Hreg(reg0_HREG);
108   HR_Writings.HR_1 = mb.Hreg(reg1_HREG);
109   HR_Writings.HR_2 = mb.Hreg(reg2_HREG);
110
111   // Send message via ESP-NOW
112   esp_err_t result = esp_now_send(MACAddr, (uint8_t *) &HR_Writings,
      ↪  sizeof(HR_Writings));
113
114   if (result == ESP_OK) {
115     Serial.println("Sent with success");
116   }
117   else {
118     Serial.println("Error sending the data");
119   }
120
121   delay(6000);
122 }
```

Apêndice B

Código do Microcontrolador ESP32-02

Este apêndice contém as linhas de código do microcrontrolador ESP32-02 para configuração como dispositivo ESP-NOW receptor para receber os dados do microcontrolador ESP32-01 via protocolo ESP-NOW, além da arquitetura NovaGenesis embarcada (EPGS) para permitir o tratamento e encapsulamento dos dados recebidos em mensagens NG para seu envio ao computador de monitoramento remoto. O código foi programado em linguagem C++ no software Eclipse IDE 2021-03.

O apêndice foi citado na Seção 4.11.

B.1 Código do arquivo main.c

```
#include <stdlib.h>
#include "esp_netif.h"
#include "esp_now.h"
#include "sdkconfig.h"

#include "runepgs.h"
#include <stdio.h>
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"

#include "esp_wifi.h"
#include "esp_private/wifi.h"
#include "esp_log.h"
#include "esp_system.h"
#include "esp_event.h"
#include "nvs_flash.h"
```

```
19 static const char* TAG = "epgs-end-node";
20 static bool s_sta_is_connected = false;
21 static bool wifi_is_connected = false;
22 #define CONFIG_EXAMPLE_WIFI_SSID "OpenWrt"
23 #define CONFIG_EXAMPLE_WIFI_PASSWORD "ng#gandalf*2016"
24
25 // process loop
26
27 void task_processloop(void *p)
28 {
29         uint32_t tick_time = 0;
30
31         // call startepgs() [runepgs.c]
32
33         startepgs();
34         tick_time = xTaskGetTickCount();
35
36         for(;;) {
37
38                 // call mydatarcv()
39
40                 my_data_receive(&HR_Data);
41
42                 if( (xTaskGetTickCount() - tick_time) >=
                    ↪  (4000/portTICK_PERIOD_MS) ) {
43                         tick_time = xTaskGetTickCount();
44
45                         // call runepgs() [runepgs.]
46
47                         runepgs();
48                         ets_printf("Running EPGS routine /// In state: ");
49
50                         // Switch Case NG State
51
52                         switch(tagNgEPGS->ngState)
53                         {
54                         case 1:
55                             ets_printf("HELLO");
56                             break;
57                         case 2:
58                             ets_printf("WAIT_HELLO_PGCS");
59                             break;
60                         case 3:
```

```
61                              ets_printf("EXPOSITION");
62                              break;
63                          case 4:
64                              ets_printf("SERVICE_OFFER");
65                              break;
66                          case 5:
67                              ets_printf("WAIT_SERVICE_ACCEPTANCE_NOTIFY");
68                              break;
69                          case 6:
70                              ets_printf("SUBSCRIBE_SERVICE_ACCEPTANCE");
71                              break;
72                          case 7:
73                              ets_printf("WAIT_SERVICE_ACCEPTANCE_DELIVERY");
74                              break;
75                          case 8:
76                              ets_printf("PUB_DATA");
77                              break;
78                          }
79                          ets_printf("\n");
80                  }
81                  vTaskDelay(1);
82          }
83  }
84
85  // rcv_cb
86
87  void recv_cb(const uint8_t *mac_addr, const uint8_t *HR_Readings, int len)
88  {
89
90          // receive and store Data
91
92          memcpy(&HR_Data, HR_Readings, len);
93
94          vTaskDelay(4000/portTICK_PERIOD_MS);
95  }
96
97  // mydatarcv()
98
99  void my_data_receive(const struct_message *HR_Data)
100 {
101
102         //put data on variables
103
```

```
104         Hold_Reg_0 = HR_Data->HR_0;
105         Hold_Reg_1 = HR_Data->HR_1;
106         Hold_Reg_2 = HR_Data->HR_2;
107 }
108
109 // Event handler for Wi-Fi
110 static void wifi_event_handler(void *arg, esp_event_base_t event_base,
111                                int32_t event_id, void *event_data)
112 {
113     switch (event_id) {
114     case SYSTEM_EVENT_STA_START:
115         printf("SYSTEM_EVENT_STA_START\r\n");
116         break;
117
118     case SYSTEM_EVENT_STA_CONNECTED:
119         printf("SYSTEM_EVENT_STA_CONNECTED\r\n");
120         wifi_is_connected = true;
121
122         esp_wifi_internal_reg_rxcb(ESP_IF_WIFI_STA,
            ↪  (wifi_rxcb_t)wifi_rx_msg);
123         break;
124
125     case SYSTEM_EVENT_STA_GOT_IP:
126         printf("SYSTEM_EVENT_STA_GOT_IP\r\n");
127         break;
128
129     case SYSTEM_EVENT_STA_DISCONNECTED:
130         printf("SYSTEM_EVENT_STA_DISCONNECTED\r\n");
131         wifi_is_connected = false;
132         esp_wifi_internal_reg_rxcb(ESP_IF_WIFI_STA, NULL);
133         esp_wifi_connect();
134         break;
135     case WIFI_EVENT_AP_STACONNECTED:
136         ESP_LOGI(TAG, "Wi-Fi AP got a station connected");
137         s_sta_is_connected = true;
138         esp_wifi_internal_reg_rxcb(ESP_IF_WIFI_AP,
            ↪  (wifi_rxcb_t)wifi_rx_msg);
139         break;
140     case WIFI_EVENT_AP_STADISCONNECTED:
141         ESP_LOGI(TAG, "Wi-Fi AP got a station disconnected");
142         s_sta_is_connected = false;
143         esp_wifi_internal_reg_rxcb(ESP_IF_WIFI_AP, NULL);
144         break;
```

```
145     default:
146         break;
147     }
148 }
149 
150 static void initialize_wifi(void)
151 {
152         // initialize wifi
153 
154         ESP_ERROR_CHECK(esp_event_loop_create_default());
155     ESP_ERROR_CHECK(esp_event_handler_register(WIFI_EVENT, ESP_EVENT_ANY_ID,
        ↪  wifi_event_handler, NULL));
156     wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
157     ESP_ERROR_CHECK(esp_wifi_init(&cfg));
158     ESP_ERROR_CHECK(tcpip_adapter_clear_default_wifi_handlers());
159     ESP_ERROR_CHECK(esp_wifi_set_storage(WIFI_STORAGE_RAM));
160         ESP_ERROR_CHECK(esp_wifi_set_ps(WIFI_PS_NONE));
161 
162     wifi_config_t wifi_config = {
163                     .sta = {
164                                         .ssid = CONFIG_EXAMPLE_WIFI_SSID,
165                                             .password =
                                                ↪  CONFIG_EXAMPLE_WIFI_PASSWORD
166                     },
167     };
168     ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
169     ESP_ERROR_CHECK(esp_wifi_set_config(ESP_IF_WIFI_STA, &wifi_config));
170     ESP_ERROR_CHECK(esp_wifi_start());
171     ESP_ERROR_CHECK(esp_wifi_connect());
172 
173     ESP_LOGI(TAG, "wifi_init_sta finished.");
174     ESP_LOGI(TAG, "connect to ap SSID:\%s password:\%s",
175                      CONFIG_EXAMPLE_WIFI_SSID, CONFIG_EXAMPLE_WIFI_PASSWORD);
176 
177         // initialize ESP-NOW
178 
179         ESP_ERROR_CHECK( esp_now_init() );
180 
181         // registes rcv_cb
182 
183     ESP_ERROR_CHECK( esp_now_register_recv_cb(recv_cb) );
184 }
185 
```

```
186 void app_main()
187 {
188 
189         // NVS Partition
190 
191         esp_err_t ret = nvs_flash_init();
192         if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret ==
            ↪  ESP_ERR_NVS_NEW_VERSION_FOUND) {
193                 ESP_ERROR_CHECK(nvs_flash_erase());
194                 ret = nvs_flash_init();
195         }
196         ESP_ERROR_CHECK(ret);
197 
198         // initialize wifi
199 
200         initialize_wifi();
201 
202         // process loop
203 
204         xTaskCreatePinnedToCore(&task_processloop, "task_processloop",
            ↪  60480, NULL, 5, task_processloop_handle, 1);
205 
206 
207 }
```

B.2 Código do arquivo runegs.c

```
1  /**
2   * Programa de Testes do EPGS
3   */
4 
5  #include "epgs_wrapper.h"
6  #include "Common\ng_hash_table.h"
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include "runepgs.h"
10 #include <string.h>
11 #include "esp_log.h"
12 
13 const char ucIdentify[] = {"24:6f:28:aa:b9:68"};
14 const char ucHID[] = {"12345"};
15 const char ucSOID[] = {"NG_SO"};
```

```
16 const char ucPID[] = {"4321"};
17
18 const char ucStack[] = {"Wi-Fi"};
19 const char ucInterface[] = {"eth1"};
20 const char ucPLCDeviceType[] = {"OutputDevices"};
21
22 void startepgs()
23 {
24         // call InitEPGS()
25
26         initEPGS(&tagNgEPGS);
27
28         // call setHwCfg()
29
30         setHwConfigurations(&tagNgEPGS, ucHID, ucSOID, ucStack, ucInterface,
           ↪  ucIdentify);
31
32         // call +keywords()
33
34         addKeyWords(&tagNgEPGS, ucPLCDeviceType);
35
36         // call +Features()
37
38         addHwSensorFeature(&tagNgEPGS, "PLCDeviceType", "OutputDevices");
39         addHwSensorFeature(&tagNgEPGS, "MotorOutput", "Q0");
40         addHwSensorFeature(&tagNgEPGS, "PistonOutput", "Q1");
41         addHwSensorFeature(&tagNgEPGS, "LampOutput", "Q2");
42         addHwSensorFeature(&tagNgEPGS, "OutputON", "0");
43         addHwSensorFeature(&tagNgEPGS, "OutputOFF", "1");
44
45 }
46
47 void runepgs(void)
48 {
49
50         //call processloop()
51
52         processLoop(&tagNgEPGS);
53 }
54
55 void wifi_tx_msg(void *buffer, uint16_t len)
56 {
```

```
57          esp_wifi_internal_tx(ESP_IF_WIFI_STA, buffer,
            ↪  len);        //        Transmit the msg through wifi
58  }
59
60  // Forward packets from Wi-Fi to Ethernet
61  esp_err_t wifi_rx_msg(void *buffer, uint16_t len, void *eb)
62  {
63          if(((char*)buffer)[12]==0x12 && ((char*)buffer)[13]==0x34)
64          {
65                  printf("\n WiFi Received(size = \%u): ", len);
66                  int i;
67                  if(((char*)buffer)[21]==0)
68                  {
69                          for(i=0;i<30;i++)
70                          {
71                                  printf(" \%02X", (unsigned
                                    ↪  int)((char*)buffer)[i]);
72                          }
73                  }
74                  else
75                  {
76                          for(i=0;i<22;i++)
77                          {
78                                  printf(" \%02X", (unsigned
                                    ↪  int)((char*)buffer)[i]);
79                          }
80                  }
81                  printf(" ");
82                  for(;i<len;i++)
83                  {
84                          printf("\%c", ((char*)buffer)[i]);
85                  }
86                  printf("\n");
87                  // NG PROCESSA A MENSAGEM
88                  newReceivedMessage(&tagNgEPGS, (char*)buffer, len);
89          }
90          else
91                  printf(".");
92
93      esp_wifi_internal_free_rx_buffer(eb);
94      return ESP_OK;
95  }
```

B.3 Biblioteca runepgs.h

```
/*
 * runepgs.h
 *
 *  Created on: May 19, 2018
 *      Author: Unknown
 */

#ifndef RUNEPGS_H_
#define RUNEPGS_H_
#include "epgs.h"
#include "DataStructures/epgs_structures.h"
#include "epgs_wrapper.h"
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/semphr.h"
#include "freertos/task.h"
#include "esp_wifi.h"
#include "esp_private/wifi.h"

typedef struct{
  uint8_t buf[257];
  uint8_t len;
} msg_packet;

NgEPGS* tagNgEPGS;

TaskHandle_t task_processloop_handle;

typedef struct struct_message {
    int HR_0;
    int HR_1;
        int HR_2;
} struct_message;

struct_message HR_Data;

int Hold_Reg_0;
int Hold_Reg_1;
int Hold_Reg_2;

void my_data_receive(const struct_message *HR_Data);
```

```
42
43 int Count;
44
45 char *dataP;
46
47 void startepgs();
48
49 void runepgs(void);
50
51 void wifi_tx_msg(void *buffer, uint16_t len);
52 esp_err_t wifi_rx_msg(void *buffer, uint16_t len, void *eb);
53
54 uint8_t seq; // NG
55
56 uint8_t TXPermission; //NG
57
58 uint8_t cont;
59
60 #endif /* RUNEPGS_H_ */
```

B.4 Código do arquivo epgs.c

```
1  /*
2  ============================================================================
3   Name        : EPGS.c
4   Author      : £(author)
5   Version     :
6   Copyright   : £(copyright)
7   Description : main definition
8  ============================================================================
9   */
10
11 #include "epgs.h"
12 #include "epgs_wrapper.h"
13 #include "Common/ng_epgs_hash.h"
14 #include "Controller/epgs_controller.h"
15 #include "Network/PG.h"
16 #include "freertos/FreeRTOS.h"
17 #include "freertos/task.h"
18 #include "runepgs.h"
19 #include "string.h"
20
```

```
21 // initEPGS()
22
23 int initEPGS(NgEPGS **ngEPGS) {
24
25         if(*ngEPGS) {
26                 return NG_ERROR;
27         }
28
29         (*ngEPGS) = (NgEPGS*) ng_malloc(sizeof(NgEPGS));
30
31         (*ngEPGS)->NetInfo = (NgNetInfo*) ng_malloc(sizeof(NgNetInfo));
32
33         (*ngEPGS)->PGCSNetInfo = NULL;
34
35         (*ngEPGS)->PGCSScnIDInfo = NULL;
36
37         (*ngEPGS)->PSSScnIDInfo = NULL;
38
39         (*ngEPGS)->APPScnIDInfo = NULL;
40
41         (*ngEPGS)->ReceivedMsg = NULL;
42
43         (*ngEPGS)->HwDescriptor = (NgHwDescriptor*) ng_malloc
        ↪  (sizeof(NgHwDescriptor));
44         (*ngEPGS)->HwDescriptor->keyWords = NULL;
45         (*ngEPGS)->HwDescriptor->keyWordsCounter = 0;
46
47         (*ngEPGS)->HwDescriptor->sensorFeatureName = NULL;
48         (*ngEPGS)->HwDescriptor->sensorFeatureValue = NULL;
49         (*ngEPGS)->HwDescriptor->featureCounter = 0;
50
51         addKeyWords(ngEPGS, "EPGS");
52         addKeyWords(ngEPGS, "Embedded_Proxy_Gateway_Service");
53
54         (*ngEPGS)->MessageCounter = 0;
55         (*ngEPGS)->ngState = HELLO;
56         (*ngEPGS)->HelloCounter = 0;
57
58         (*ngEPGS)->pubDataFileName = NULL;
59         (*ngEPGS)->pubDataSize = 0;
60         (*ngEPGS)->pubData = NULL;
61
62         (*ngEPGS)->key = NULL;
```

```
63
64          return NG_OK;
65  }
66
67  int destroyEPGS(NgEPGS **ngEPGS) {
68          destroy_NgEPGS(ngEPGS);
69          return 1;
70  }
71
72  // setHwCfg()
73
74  int setHwConfigurations(NgEPGS **ngEPGS, const char* hid, const char* soid,
    ↪  const char* Stack, const char* Interface, const char* Identifier) {
75          if(!(*ngEPGS)->NetInfo) {
76                          return NG_ERROR;
77                  }
78
79                  ng_strcpy((*ngEPGS)->NetInfo->HID, hid);
80                  ng_strcpy((*ngEPGS)->NetInfo->SOID, soid);
81
82                  (*ngEPGS)->NetInfo->Stack = (char*) ng_malloc(sizeof(char) *
                    ↪  (ng_strlen(Stack) + 1));
83                  ng_strcpy((*ngEPGS)->NetInfo->Stack, Stack);
84
85                  (*ngEPGS)->NetInfo->Interface = (char*)
                    ↪  ng_malloc(sizeof(char) * (ng_strlen(Interface) + 1));
86                  ng_strcpy((*ngEPGS)->NetInfo->Interface, Interface);
87
88                  (*ngEPGS)->NetInfo->Identifier = (char*)
                    ↪  ng_malloc(sizeof(char) * (ng_strlen(Identifier) + 1));
89                  ng_strcpy((*ngEPGS)->NetInfo->Identifier, Identifier);
90
91                  return NG_OK;
92  }
93
94  // processloop()
95
96  int processLoop(NgEPGS **ngEPGS) {
97
98
99          char ucFile[100];
100         char ucBufferName[100];
101         switch((*ngEPGS)->ngState) {
```

```
102                 case HELLO:
103                         if((*ngEPGS)->HelloCounter >= 5 ) {
104                                 (*ngEPGS)->ngState = WAIT_HELLO_PGCS;
105                                 (*ngEPGS)->HelloCounter = 0;
106                         }
107                         (*ngEPGS)->HelloCounter++;
108
109                         // call RunHello()
110
111                         RunHello((*ngEPGS));
112                         if((*ngEPGS)->PGCSNetInfo &&
                            ↪  (*ngEPGS)->PGCSScnIDInfo &&
                            ↪  (*ngEPGS)->PSSScnIDInfo) {
113                             (*ngEPGS)->ngState = EXPOSITION;
114                         }
115                         break;
116
117                 case WAIT_HELLO_PGCS:
118
119                         // Wait PGCS info
120
121                         break;
122
123                 case EXPOSITION:
124
125                         // call RunExpo()
126
127                         if(RunExposition((*ngEPGS)) == NG_OK) {
128                                 (*ngEPGS)->ngState = SERVICE_OFFER;
129                                 cont = 0;
130                         }
131                         break;
132
133                 case SERVICE_OFFER:
134
135                         if (cont < 5) {
136
137                                 // call RubPubServiceOffer()
138
139                                 if(RunPubServiceOffer((*ngEPGS)) == NG_OK) {
140                                         (*ngEPGS)->ngState =
                                         ↪  WAIT_SERVICE_ACCEPTANCE_NOTIFY;
141                                         cont++;
```

```
142                         }
143                 }
144                 break;
145
146         case WAIT_SERVICE_ACCEPTANCE_NOTIFY:
147
148                 if (cont < 5) {
149                         (*ngEPGS)->ngState = SERVICE_OFFER;
150                 }
151                 //IDLE TO RECEIVE A POSSIBLE SERVICE ACCEPTANCE
                    ↪ RESPONSE
152                 //NG ONLY LEAVES THIS STATE WHEN IT RECEIVES A
                    ↪ SERVICE ACCEPTANCE NOTIFY MESSAGE
153                 break;
154
155         case SUBSCRIBE_SERVICE_ACCEPTANCE:
156
157                 if (cont < 10) {
158
159                         // call RunSubServiceAcc()
160
161                         if(RunSubscribeServiceAcceptance((*ngEPGS))
                            ↪ == NG_OK) {
162                                 (*ngEPGS)->ngState =
                                    ↪ WAIT_SERVICE_ACCEPTANCE_DELIVERY;
163                                 cont++;
164                         }
165                 }
166                 break;
167
168         case WAIT_SERVICE_ACCEPTANCE_DELIVERY:
169
170                 if (cont < 10) {
171                         (*ngEPGS)->ngState =
                            ↪ SUBSCRIBE_SERVICE_ACCEPTANCE;
172                 }
173                 //NG ONLY LEAVES THIS STATE WHEN IT RECEIVES
                    ↪ SERVICE ACCEPTANCE DELIVERY MESSAGE WITH NO
                    ↪ ERRORS
174                 break;
175
176         case PUB_DATA:
177
```

```
178                         sprintf(ucFile, "{ Data: \%i,\%i,\%i }", Hold_Reg_0,
                            ↪  Hold_Reg_1, Hold_Reg_2);
179                         sprintf(ucBufferName,"Measures_\%s_\%d.json",
                            ↪  tagNgEPGS->NetInfo->HID, Count);
180                         setDataToPub(&tagNgEPGS, ucBufferName, ucFile,
                            ↪  strlen(ucFile));
181                         Count++;
182                         break;
183         }
184
185         return NG_OK;
186 }
187
188
189
190 int newReceivedMessage(NgEPGS **ngEPGS, const char *message, int msgSize) {
191         int result = newMessageReceived (ngEPGS, message, msgSize);
192
193         if(result != NG_PROCESSING) {
194 //                  processLoop(ngEPGS); //Let the periodic processLoop in
    ↪  runEPGS handle it
195                 destroy_NgReceivedMsg(&((*ngEPGS)->ReceivedMsg));
196         }
197
198         return result;
199 }
200
201 // +keywords()
202
203 int addKeyWords(NgEPGS **ngEPGS, const char* key) {
204         (*ngEPGS)->HwDescriptor->keyWords = (char**)
            ↪  ng_realloc((*ngEPGS)->HwDescriptor->keyWords,
            ↪  ((*ngEPGS)->HwDescriptor->keyWordsCounter + 1) * sizeof(char*));
205         (*ngEPGS)->HwDescriptor->keyWords[(*ngEPGS)->HwDescriptor->keyWordsCounter]
            ↪  = (char*)ng_malloc(sizeof(char) * (ng_strlen(key) + 1));
206         ng_strcpy((*ngEPGS)->HwDescriptor->keyWords[(*ngEPGS)->HwDescriptor
            ↪  ->keyWordsCounter], key);
207
208         (*ngEPGS)->HwDescriptor->keyWordsCounter++;
209
210         return NG_OK;
211 }
212
```

```
213 // +Features()
214
215 int addHwSensorFeature(NgEPGS **ngEPGS, const char* name, const char* value)
    ↪  {
216         (*ngEPGS)->HwDescriptor->sensorFeatureName = (char**)
            ↪  ng_realloc((*ngEPGS)->HwDescriptor->sensorFeatureName,
            ↪  ((*ngEPGS)->HwDescriptor->featureCounter + 1) * sizeof(char*));
217         (*ngEPGS)->HwDescriptor->sensorFeatureName[(*ngEPGS)->HwDescriptor
            ↪  ->featureCounter] = (char*)ng_malloc(sizeof(char) *
            ↪  (ng_strlen(name) + 1));
218         ng_strcpy((*ngEPGS)->HwDescriptor->sensorFeatureName[(*ngEPGS)->
            ↪  HwDescriptor->featureCounter], name);
219
220         (*ngEPGS)->HwDescriptor->sensorFeatureValue = (char**)
            ↪  ng_realloc((*ngEPGS)->HwDescriptor->sensorFeatureValue,
            ↪  ((*ngEPGS)->HwDescriptor->featureCounter + 1) * sizeof(char*));
221         (*ngEPGS)->HwDescriptor->sensorFeatureValue[(*ngEPGS)->HwDescriptor->
            ↪  featureCounter] = (char*)ng_malloc(sizeof(char) *
            ↪  (ng_strlen(value) + 1));
222         ng_strcpy((*ngEPGS)->HwDescriptor->sensorFeatureValue[(*ngEPGS)->
            ↪  HwDescriptor->featureCounter], value);
223
224         (*ngEPGS)->HwDescriptor->featureCounter++;
225
226         return NG_OK;
227 }
228
229 // setDataPub()
230
231 int setDataToPub(NgEPGS **ngEPGS, const char* fileName, const char* data,
    ↪  int dataLength) {
232
233         if(!fileName || !data || dataLength <= 0) {
234                 return NG_ERROR;
235         }
236
237         (*ngEPGS)->pubDataFileName = (char*)
            ↪  ng_realloc((*ngEPGS)->pubDataFileName, (ng_strlen(fileName) + 1)
            ↪  * sizeof(char));
238         (*ngEPGS)->pubData = (char*) ng_realloc((*ngEPGS)->pubData,
            ↪  (dataLength) * sizeof(char));
239
240         ng_strcpy((*ngEPGS)->pubDataFileName, fileName);
```

```
241
242         int i = 0;
243         for(i = 0; i<dataLength; i++) {
244                 (*ngEPGS)->pubData[i] = data[i];
245         }
246
247         (*ngEPGS)->pubDataSize = dataLength;
248
249         // call RunPubData()
250
251         if((*ngEPGS)->pubData) {
252                 RunPublishData((*ngEPGS));
253         }
254
255         return NG_OK;
256 }
```

B.5 Biblioteca epgs.h

```
1  #ifndef NG_EPGS_H_
2  #define NG_EPGS_H_
3
4  #include "epgs_structures.h"
5
6  /**
7   *
8   * Inits the EPGS data structure, flags and counters.
9   *
10  * @param ngEPGS Reference to the EPGS data structure pointer.
11  *
12  * @return OK if success, ERROR otherwise (@see epgs_defines.h)
13  */
14 int initEPGS(NgEPGS **ngEPGS);
15
16 /**
17  * Sets the Hardware configuration
18  *
19  * @param ngEPGS                Reference to the EPGS data structure
    ↪  pointer.
20  * @param hid                       The Hardware Id - Unique number that
    ↪  describes the hardware
21  * @param soid                      System Operation Id
```

```
22  * @param pid                          Process Id
23  * @param Stack                        Type of network stack used (eg:
    ↪  wi-fi, ethernet, bluetooth, ip-v6...)
24  * @param Interface                    Interface of the network (eg: eth0,
    ↪  em0...)
25  * @param Identifier         The identifier of the network (eg:
    ↪  00:12:34:56:78:9A)
26  *
27  * @return OK if success, ERROR otherwise (@see epgs_defines.h)
28  */
29 int setHwConfigurations(NgEPGS **ngEPGS, const char* hid, const char* soid,
    ↪  const char* Stack, const char* Interface, const char* Identifier);
30
31 /**
32  * Adds keywords that specifies the Node
33  *
34  * @param ngEPGS                  Reference to the EPGS data structure
    ↪  pointer.
35  * @param name                         The keyword (eg: "Temperature").
36  *
37  * @return OK if success, ERROR otherwise (@see epgs_defines.h)
38  */
39 int addKeyWords(NgEPGS **ngEPGS, const char* name);
40
41 /**
42  * Adds sensor features in name/value way.
43  *
44  * @param ngEPGS                  Reference to the EPGS data structure
    ↪  pointer.
45  * @param name                         The name of the feature (eg:
    ↪  Temperature_Max).
46  * @param value                         The value of the feature (eg: 100).
47  *
48  * @return OK if success, ERROR otherwise (@see epgs_defines.h)
49  */
50 int addHwSensorFeature(NgEPGS **ngEPGS, const char* name, const char*
    ↪  value);
51
52 /**
53  * Analysis the EPGS state and takes actions. This function must be called
    ↪  in a loop.
54  *
```

```
55  * @param ngEPGS                  Reference to the EPGS data structure
    ↪  pointer.
56  *
57  * @return State Machine ID (@see epgs_structures.h)
58  */
59 int processLoop(NgEPGS **ngEPGS);
60
61 /**
62  * Parses the received message and takes actions.
63  *
64  * @param ngEPGS                  Reference to the EPGS data structure
    ↪  pointer.
65  * @param message                 The received message.
66  * @param msgSize                 The size of the received message.
67  *
68  * @return OK if success, ERROR otherwise (@see epgs_defines.h)
69  */
70 int newReceivedMessage(NgEPGS **ngEPGS, const char *message, int msgSize);
71
72 /**
73  * Sets the data to be publish in the next cycle.
74  *
75  * @param ngEPGS                  Reference to the EPGS data structure
    ↪  pointer.
76  * @param fileName                 The name of file to be published.
77  * @param data                        Data content of the file to be
    ↪  published.
78  * @param dataLengh                 Size of the data to be published.
79  *
80  * @return OK if success, ERROR otherwise (@see epgs_defines.h)
81  */
82 int setDataToPub(NgEPGS **ngEPGS, const char* fileName, const char* data,
    ↪  int dataLength);
83
84
85 #endif /* NG_EPGS_H_ */
```

B.6 Código do arquivo epgs_controller.c

```
1 #include "epgs_controller.h"
2 #include "epgs_structures.h"
3 #include "epgs_wrapper.h"
```

```
4  #include "epgs_hello.h"
5  #include "epgs_pubNotify.h"
6  #include "epgs_subServiceAcceptance.h"
7  #include "epgs_exposition.h"
8  #include "epgs_defines.h"
9  #include "ng_command.h"
10 #include "ng_message.h"
11 #include "ng_util.h"
12 #include "ng_json.h"
13 #include "PG.h"
14 #include "runepgs.h"
15
16 // RunHello()
17
18 int RunHello(NgEPGS *ngEPGS) {
19
20         if(!ngEPGS) {
21                 return NG_ERROR;
22         }
23         if(!(ngEPGS->NetInfo)) {
24                 return NG_ERROR;
25         }
26
27         NgMessage *helloMessage = NULL;
28
29         // call Action1()
30
31         ActionRunHello(ngEPGS->NetInfo, &helloMessage);
32
33         // call SendNGMsg()
34
35         sendNGMessage(ngEPGS, helloMessage, true);
36
37         ng_destroy_message(&helloMessage);
38
39         return NG_OK;
40 }
41
42 // RunExpo()
43
44 int RunExposition(NgEPGS *ngEPGS) {
45         int result = NG_ERROR;
46
```

```
47          if(!ngEPGS) {
48                  return result;
49          }
50          if(!ngEPGS->NetInfo) {
51                  return result;
52          }
53          if(!ngEPGS->PSSScnIDInfo) {
54                  return result;
55          }
56
57          NgMessage *expositioneMessage = NULL;
58
59          // call Action2()
60
61          result = actionExpostion(ngEPGS, &expositioneMessage);
62
63          if(result == NG_OK) {
64
65                  // call SendNGMsg()
66
67                  result = sendNGMessage(ngEPGS, expositioneMessage, false);
68          }
69          ng_destroy_message(&expositioneMessage);
70
71          return result;
72  }
73
74  // RunPubServiceOffer()
75
76  int RunPubServiceOffer(NgEPGS *ngEPGS) {
77          if(!ngEPGS) {
78                  return NG_ERROR;
79          }
80          if(!ngEPGS->NetInfo) {
81                  return NG_ERROR;
82          }
83          if(!ngEPGS->PGCSNetInfo) {
84                  return NG_ERROR;
85          }
86          if(!ngEPGS->PGCSScnIDInfo) {
87                  return NG_ERROR;
88          }
89          if(!ngEPGS->PSSScnIDInfo) {
```

```
90                  return NG_ERROR;
91          }
92
93          char* filePayload = (char*)ng_calloc(sizeof(char), 600);
94          ng_strcpy(filePayload, "ng -sr --b 0.1 [ < 1 s 17 > < 1 s
            ↪  EPGS_PLCOutputDevices > ");
95
96          ng_sprintf(filePayload + ng_strlen(filePayload), "< \%i s ",
            ↪  ngEPGS->HwDescriptor->featureCounter);
97          int i = 0;
98          for(i = 0; i < ngEPGS->HwDescriptor->featureCounter; i++) {
99                  ng_sprintf(filePayload + ng_strlen(filePayload), "\%s ",
                    ↪  ngEPGS->HwDescriptor->sensorFeatureName[i]);
100         }
101         ng_sprintf(filePayload + ng_strlen(filePayload), "> ]\n");
102
103         for(i = 0; i < ngEPGS->HwDescriptor->featureCounter; i++) {
104                 ng_sprintf(filePayload + ng_strlen(filePayload), "ng -sr --b
                    ↪  0.1 [ < 1 s 17 > < 1 s \%s > < 1 s \%s > ]\n",
                    ↪  ngEPGS->HwDescriptor->sensorFeatureName[i],
                    ↪  ngEPGS->HwDescriptor->sensorFeatureValue[i]);
105         }
106
107         NgMessage *pubServiceOfferMessage = NULL;
108
109         // call Action3()
110
111         actionPublicationAndNotification(ngEPGS, false, "Service_Offer.txt",
            ↪  filePayload, (unsigned long long)ng_strlen(filePayload),
            ↪  &pubServiceOfferMessage);
112         ng_free(filePayload);
113
114         // call SendNGMsg()
115
116         sendNGMessage(ngEPGS, pubServiceOfferMessage, false);
117
118         ng_destroy_message(&pubServiceOfferMessage);
119
120         return NG_OK;
121
122 }
123
124 // RunSubServiceAcc()
```

```
125
126 int RunSubscribeServiceAcceptance(NgEPGS *ngEPGS) {
127         int result = NG_ERROR;
128
129         if(!ngEPGS) {
130                 return result;
131         }
132         if(!ngEPGS->NetInfo) {
133                 return result;
134         }
135         if(!ngEPGS->PGCSScnIDInfo) {
136                 return result;
137         }
138         if(!ngEPGS->PSSScnIDInfo) {
139                 return result;
140         }
141
142         NgMessage *subServiceAcceptanceMessage = NULL;
143
144         // call Action4()
145
146         result = ActionSubscriptionServiceAcceptance(ngEPGS->NetInfo,
            ↪  ngEPGS->PGCSScnIDInfo, ngEPGS->PSSScnIDInfo,
            ↪  ngEPGS->MessageCounter, ngEPGS->key,
            ↪  &subServiceAcceptanceMessage);
147
148         if(result == NG_OK) {
149
150                 // call SendNGMsg()
151
152                 result = sendNGMessage(ngEPGS, subServiceAcceptanceMessage,
                    ↪  false);
153         }
154         ng_destroy_message(&subServiceAcceptanceMessage);
155
156         return result;
157 }
158
159 // RunPubData
160
161 int RunPublishData(NgEPGS *ngEPGS) {
162         if(!ngEPGS) {
163                 return NG_ERROR;
```

```
164         }
165         if(!ngEPGS->PGCSNetInfo) {
166                 return NG_ERROR;
167         }
168         if(!ngEPGS->PGCSScnIDInfo) {
169                 return NG_ERROR;
170         }
171         if(!ngEPGS->PSSScnIDInfo) {
172                 return NG_ERROR;
173         }
174         if(!ngEPGS->pubData) {
175                 return NG_ERROR;
176         }
177
178         NgMessage *pubDataMessage = NULL;
179
180         // call Action3()
181
182         actionPublicationAndNotification(ngEPGS, true,
            ↪  ngEPGS->pubDataFileName, ngEPGS->pubData, ngEPGS->pubDataSize,
            ↪  &pubDataMessage);
183
184         // call SendNGMsg()
185
186         sendNGMessage(ngEPGS, pubDataMessage, false);
187         ng_destroy_message(&pubDataMessage);
188
189         return NG_OK;
190
191 }
192
193 // ParseRcvMsg()
194
195 int ParseReceivedMessage(NgEPGS **ngEPGS) {
196
197         int result = NG_OK;
198         bool updateSCNs = false;
199
200         NgPGCSInfo* pgcsInfo = NULL;
201         NgScnIDInfo* pgcsSCNInfo = NULL;
202         NgScnIDInfo* pssSCNInfo = NULL;
203
204
```

```
//          ets_printf("\n<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-")
//          ets_printf("\n NOVAGENESIS MSG RECEIVED:");
//          ets_printf("\n\%s", (*ngEPGS)->ReceivedMsg->buffer);
//          ets_printf("\n<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-")

        NgMessage *ngMessage;
        MessageFromString((*ngEPGS)->ReceivedMsg->buffer, (*ngEPGS)->ReceivedMsg->mgs_size, &ngMessage);


        int i = 0;
        for(i = 0; i< ngMessage->NoCL; i++) {
                NgCommand *CL = NULL;
                GetCommandLine(ngMessage, i, &CL);

                if(ng_strcmp("-m",CL->Name)==0 && ng_strcmp("--cl",CL->Alternative)==0) {

                        if(CL->NoA == 3) {
                                unsigned int nEle;
                                GetNumberofArgumentElements(CL, 1, &nEle);
                                if(nEle == 4 ){
                                        pgcsSCNInfo = (NgScnIDInfo*) ng_malloc(sizeof(NgScnIDInfo));

                                        char *value;

                                        GetArgumentElement(CL, 1, 0, &value);
                                        ng_strcpy(pgcsSCNInfo->HID, value);
                                        ng_free(value);

                                        GetArgumentElement(CL, 1, 1, &value);
                                        ng_strcpy(pgcsSCNInfo->OSID, value);
                                        ng_free(value);

                                        GetArgumentElement(CL, 1, 2, &value);
                                        ng_strcpy(pgcsSCNInfo->PID, value);
                                        ng_free(value);

```

```
241                                     GetArgumentElement(CL, 1, 3,
                                        ↪ &value);
242                                     ng_strcpy(pgcsSCNInfo->BID, value);
243                                     ng_free(value);
244                             }
245                              else {
246                             result = NG_ERROR;
247                             }
248                     } else {
249                             result = NG_ERROR;
250                     }
251             }
252
253             else if(ng_strcmp("-hello",CL->Name)==0 &&
                ↪ ng_strcmp("--ihc",CL->Alternative)==0 &&
                ↪ ng_strcmp("0.2",CL->Version)==0) {
254
255                     if(CL->NoA == 2) {
256                             unsigned int nEle;
257                             GetNumberofArgumentElements(CL, 0, &nEle);
258
259                             if(nEle == 6 ){
260                                     pgcsInfo = (NgPGCSInfo*)
                                        ↪ ng_malloc(sizeof(NgPGCSInfo));
261
262                                     char *value;
263
264                                     GetArgumentElement(CL, 0, 0,
                                        ↪ &value);
265                                     ng_strcpy(pgcsInfo->GW_SCN, value);
266                                     ng_free(value);
267
268                                     GetArgumentElement(CL, 0, 1,
                                        ↪ &value);
269                                     ng_strcpy(pgcsInfo->HT_SCN, value);
270                                     ng_free(value);
271
272                                     GetArgumentElement(CL, 0, 2,
                                        ↪ &value);
273                                     ng_strcpy(pgcsInfo->CORE_BID_SCN,
                                        ↪ value);
274                                     ng_free(value);
275
```

```
276                                     GetArgumentElement(CL, 0, 3,
                                        ↪ &value);
277                                     pgcsInfo->Stack = (char*)
                                        ↪ ng_calloc(sizeof(char),
                                        ↪ ng_strlen(value)+1);
278                                     ng_strcpy(pgcsInfo->Stack, value);
279                                     ng_free(value);
280
281                                     GetArgumentElement(CL, 0, 4,
                                        ↪ &value);
282                                     pgcsInfo->Interface = (char*)
                                        ↪ ng_calloc(sizeof(char),
                                        ↪ ng_strlen(value)+1);
283                                     ng_strcpy(pgcsInfo->Interface,
                                        ↪ value);
284                                     ng_free(value);
285
286                                     GetArgumentElement(CL, 0, 5,
                                        ↪ &value);
287                                     pgcsInfo->Identifier = (char*)
                                        ↪ ng_calloc(sizeof(char),
                                        ↪ ng_strlen(value)+1);
288                                     ng_strcpy(pgcsInfo->Identifier,
                                        ↪ value);
289                                     ng_free(value);
290
291                                     updateSCNs = true;
292                                 }
293                                  else {
294                                 result = NG_ERROR;
295                                 }
296
297                                 GetNumberofArgumentElements(CL, 1, &nEle);
298
299                                 if(nEle == 4 ){
300                                     pssSCNInfo = (NgScnIDInfo*)
                                        ↪ ng_malloc(sizeof(NgScnIDInfo));
301
302                                     char *value;
303
304                                     GetArgumentElement(CL, 1, 0,
                                        ↪ &value);
305                                     ng_strcpy(pssSCNInfo->HID, value);
```

```
306                                     ng_free(value);
307
308                                     GetArgumentElement(CL, 1, 1,
                                        ↪  &value);
309                                     ng_strcpy(pssSCNInfo->OSID, value);
310                                     ng_free(value);
311
312                                     GetArgumentElement(CL, 1, 2,
                                        ↪  &value);
313                                     ng_strcpy(pssSCNInfo->PID, value);
314                                     ng_free(value);
315
316                                     GetArgumentElement(CL, 1, 3,
                                        ↪  &value);
317                                     ng_strcpy(pssSCNInfo->BID, value);
318                                     ng_free(value);
319                             }
320                              else {
321                             result = NG_ERROR;
322                             }
323                     } else {
324                             result = NG_ERROR;
325                     }
326
327             }
328
329             else if(ng_strcmp("-notify",CL->Name)==0 &&
                    ↪  ng_strcmp("--s",CL->Alternative)==0 ) {
330
331                     updateSCNs = false;
332
333                     if(CL->NoA == 3) {
334                             unsigned int nEle;
335                             GetNumberofArgumentElements(CL, 1, &nEle);
336
337                             if(nEle == 1 ) {
338
339                                     if((*ngEPGS)->key) {
340                                             ng_free((*ngEPGS)->key);
341                                             (*ngEPGS)->key = NULL;
342                                     }
343                                     GetArgumentElement(CL, 1, 0,
                                        ↪  &(*ngEPGS)->key);
```

```
344
345                                     if((*ngEPGS)->ngState ==
                                        ↪  WAIT_SERVICE_ACCEPTANCE_NOTIFY)(*ngEPGS)->
                                        ↪  ngState =
                                        ↪  SUBSCRIBE_SERVICE_ACCEPTANCE;
346
347
348                                     updateSCNs = false;
349                             }
350                              else {
351                                     result = NG_ERROR;
352                             }
353
354
355                     } else {
356                             result = NG_ERROR;
357                     }
358             }
359
360             else if(ng_strcmp("-d",CL->Name)==0 &&
                ↪  ng_strcmp("--b",CL->Alternative)==0 ) {
361
362                     updateSCNs = false;
363
364                     if(CL->NoA == 3) {
365                             unsigned int nEle;
366                             GetNumberofArgumentElements(CL, 1, &nEle);
367
368                             if(nEle == 1 ) {
369                                     char *key;
370                                     GetArgumentElement(CL, 1, 0, &key);
371
372                                     if(ng_strcmp((*ngEPGS)->key, key) ==
                                        ↪  0)
373                                     {
374                                             if((*ngEPGS)->APPScnIDInfo)
                                                ↪  {
375                                                     destroy_NgScnIDInfo(&(*ngEPGS)->
                                                        ↪  APPScnIDInfo);
376                                             }
377                                             (*ngEPGS)->APPScnIDInfo =
                                                ↪  (NgScnIDInfo*)
                                                ↪  ng_malloc(sizeof(NgScnIDInfo));
```

```
378
379                                     char *end_str = NULL;
380                                     char* auxStr =
                                        ↪  (char*)ng_calloc(sizeof(char),
                                        ↪  (ngMessage->PayloadSize
                                        ↪  +1));
381                                     ng_memcpy(auxStr,
                                        ↪  ngMessage->Payload,
                                        ↪  ngMessage->PayloadSize);
382
383                                     char* token =
                                        ↪  strtok_r(auxStr, " ",
                                        ↪  &end_str);
384
385                                     if(token) {
386                                             ng_strcpy((*ngEPGS)->
                                             ↪  APPScnIDInfo->HID,
                                             ↪  token);
387                                             token =
                                             ↪  strtok_r(NULL, "
                                             ↪  ", &end_str);
388                                     }
389                                     else {
390                                             result = NG_ERROR;
391                                     }
392
393                                     if(token) {
394                                             ng_strcpy((*ngEPGS)->
                                             ↪  APPScnIDInfo->OSID,
                                             ↪  token);
395                                             token =
                                             ↪  strtok_r(NULL, "
                                             ↪  ", &end_str);
396                                     }
397                                     else {
398                                             result = NG_ERROR;
399                                     }
400
401                                     if(token) {
402                                             ng_strcpy((*ngEPGS)->
                                             ↪  APPScnIDInfo->PID,
                                             ↪  token);
```

```
403                                                     token =
                                                        ↪  strtok_r(NULL, "
                                                        ↪  ", &end_str);
404                                             }
405                                             else {
406                                                     result = NG_ERROR;
407                                             }
408
409                                             if(token) {
410                                                     ng_strcpy((*ngEPGS)->
                                                        ↪  APPScnIDInfo->BID,
                                                        ↪  token);
411                                             //        ng_free(token);
412                                             }
413                                             else {
414                                                     result = NG_ERROR;
415                                             }
416
417                                             if(result == NG_OK) {
418                                                     (*ngEPGS)->ngState =
                                                        ↪  PUB_DATA;
419                                             }
420                                     }
421
422                                     ng_free(key);
423                                     updateSCNs = false;
424                             }
425                             else {
426                                     result = NG_ERROR;
427                             }
428
429
430                     } else {
431                             result = NG_ERROR;
432                     }
433             }
434             CL=NULL;
435     }
436
437     if(updateSCNs == true) {
438             if((*ngEPGS)->PGCSNetInfo) {
439                     destroy_NgPGCSInfo(&(*ngEPGS)->PGCSNetInfo);
440             }
```

```
441                 if((*ngEPGS)->PGCSScnIDInfo) {
442                         destroy_NgScnIDInfo(&(*ngEPGS)->PGCSScnIDInfo);
443                 }
444                 if((*ngEPGS)->PSSScnIDInfo) {
445                         destroy_NgScnIDInfo(&(*ngEPGS)->PSSScnIDInfo);
446                 }
447
448                 (*ngEPGS)->PGCSNetInfo = pgcsInfo;
449                 (*ngEPGS)->PGCSScnIDInfo = pgcsSCNInfo;
450                 (*ngEPGS)->PSSScnIDInfo = pssSCNInfo;
451         } else {
452
453                 if(pgcsInfo) {
454                         destroy_NgPGCSInfo(&pgcsInfo);
455                 }
456                 if(pgcsSCNInfo) {
457                         destroy_NgScnIDInfo(&pgcsSCNInfo);
458                 }
459                 if(pssSCNInfo) {
460                         destroy_NgScnIDInfo(&pssSCNInfo);
461                 }
462         }
463
464         ng_destroy_message(&ngMessage);
465
466         return result;
467 }
```

B.7 Biblioteca epgs_controller.h

```
1  /*
2   * EPGS_Controller.h
3   *
4   *  Created on: 03/02/2016
5   *      Author: vaner
6   */
7
8  #ifndef CONTROLLER_EPGS_CONTROLLER_H_
9  #define CONTROLLER_EPGS_CONTROLLER_H_
10
11 #include "EPGS.h"
12
```

```
13 int RunHello(NgEPGS *ngEPGS);
14 int RunExposition(NgEPGS *ngEPGS);
15 int RunPubServiceOffer(NgEPGS *ngEPGS);
16 int RunSubscribeServiceAcceptance(NgEPGS *ngEPGS);
17 int RunPublishData(NgEPGS *ngEPGS);
18 int ParseReceivedMessage(NgEPGS **ngEPGS);
19
20 #endif /* CONTROLLER_EPGS_CONTROLLER_H_ */
```