



UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECATRÔNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA
MECATRÔNICA



ÁLISSON CARVALHO VASCONCELOS

Sistema de Gerenciamento da Rede de Petri Inserida em RFID implementada em ESP32

Uberlândia, MG

2025

ÁLISSON CARVALHO VASCONCELOS

Sistema de Gerenciamento da Rede de Petri Inserida em RFID implementada em ESP32

Trabalho de Conclusão de Curso
apresentado à Faculdade de Engenharia
Mecânica da Universidade Federal de
Uberlândia, como requisito parcial para
obtenção do título de bacharel em Engenharia
Mecatrônica.

Área de concentração: Engenharia
Mecatrônica

Orientador: José Jean-Paul Zanlucchi
de Souza Tavares

Uberlândia, MG

2025

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

V331 Vasconcelos, Álisson Carvalho, 1994-
2025 Sistema de Gerenciamento da Rede de Petri Inserida em
RFID implementada em ESP32 [recurso eletrônico] /
Álisson Carvalho Vasconcelos. - 2025.

Orientador: José Jean-Paul Zanlucchi de Souza Tavares.
Trabalho de Conclusão de Curso (graduação) -
Universidade Federal de Uberlândia, Graduação em
Engenharia Mecatrônica.
Modo de acesso: Internet.
Inclui bibliografia.

1. Mecatrônica. I. Tavares, José Jean-Paul Zanlucchi
de Souza, 1972-, (Orient.). II. Universidade Federal de
Uberlândia. Graduação em Engenharia Mecatrônica. III.
Título.

CDU: 621.03

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:

Gizele Cristine Nunes do Couto - CRB6/2091
Nelson Marcos Ferreira - CRB6/3074

ÁLISSON CARVALHO VASCONCELOS

Sistema de Gerenciamento da Rede de Petri Inserida em RFID implementada em ESP32

Trabalho de Conclusão de Curso
apresentado à Faculdade de Engenharia
Mecânica da Universidade Federal de
Uberlândia, como requisito parcial para
obtenção do título de bacharel em Engenharia
Mecatrônica.

Área de concentração: Engenharia
Mecatrônica

Uberlândia, 14 de fevereiro de 2025

Banca Examinadora:

Prof. Dr. José Jean-Paul Zanlucchi de Souza Tavares
Universidade Federal de Uberlândia

José Jean-Paul Zanlucchi de Souza Tavares – Doutorado (USP)

Roberto Mendes Finzi – Doutorado (UFU)

Werley Rocherter Borges Ferreira – Mestrado (UFU)

AGRADECIMENTOS

Agradeço aos meus pais e meus entes querido, que servem como pilar para o meu crescimento e desenvolvimento como estudante, profissional e pessoa, me apoiando em todo os sentidos.

Agradeço ao professor Dr. José Jean-Paul Zanlucchi de Souza Tavares, que tem me orientado ativamente e com paciência, bem como pelo apoio e o incentivo para o desenvolvimento deste trabalho, fornecendo inclusive, os materiais para o seu desenvolvimento.

Agradeço ao estagiário João Gassam, que me ajudou na parte de hardware, ajudando nas correções de problemas bem como na soldagem de componentes da solução.

Agradeço aos demais frequentadores do MAPL, que de alguma forma ajudaram no desenvolvimento do projeto. Seja explicando questões importantes para o desenvolvimento ou simplesmente tendo uma conversa agradável.

Agradeço aos meus professores que durante a minha graduação me capacitaram a ter o conhecimento sobre assuntos abordados nesse projeto bem como questões externas a minha graduação.

Agradeço aos meus colegas de turma e professores que foram essenciais para a minha formação profissional e pessoal.

“A persistência é o caminho do
êxito.”

(CHARLES CHAPLIN, 1997, p. 118)

RESUMO

Em um mundo globalizado, as indústrias estão passando por um processo de transformação conhecido como indústria 4.0, uma nova revolução nas técnicas industriais, que se caracterizam pela automação industrial e pelo uso de tecnologias digitais, tais como a comunicação entre dispositivos, internet das coisas (IoT – *Internet of Things*), comunicação em nuvem, RFID (ou Identificação por Rádio Frequência do inglês *Radio Frequency Identification*) e rastreabilidade de processos. A redução do custo dos microprocessadores, juntamente com sua descentralização e distribuição em sistemas altamente complexos, tem se mostrado uma arquitetura extremamente poderosa e promissora. A complexidade e mutabilidade dos processos criam desafios na área de controle, modelagem e monitoramento de sistemas. Nesse contexto, este trabalho tem como objetivo desenvolver e implementar o Sistema de Gerenciamento da Rede de Petri Inserida em Base de Dados RFID implementada em ESP32 ou PNRD (*Petri Net Inside RFID Database* – Redes de Petri Inseridas em Base de dados RFID) *Management System implemented in ESP32* (PELMS), projetado para monitorar e rastrear componentes RFID (tags ou etiquetas e leitores) espalhados em uma planta industrial, modelados usando Redes de Petri dentro da estrutura PNRD. Usando leitores RFID instalados em equipamentos de linha de produção em uma fábrica, o sistema identificará as transições de peças ao longo do processo de fabricação por meio de Tag's anexadas a cada peça. Ele então tomará medidas para garantir que o produto atenda aos padrões de qualidade desejados, sendo capaz de detectar automaticamente quaisquer anomalias ou não conformidade no processo esperado. Tanto as Redes de Petri quanto os RFID são componentes essenciais para o monitoramento e gerenciamento da cadeia produtiva, pois permitem a rastreabilidade, identificação e padronização dos produtos. O PELMS desempenha um papel essencial no setup do sistema PNRD, bem como no monitoramento e exibição do status dos elementos da rede.

Palavras-chave: RFID, Redes de Petri, ESP32, PNRD, PELMS.

ABSTRACT

In a globalized world, industries are undergoing a transformation process known as Industry 4.0, a new revolution in industrial techniques characterized by industrial automation and the use of digital technologies such as device-to-device communication, the Internet of Things (IoT), cloud communication, RFID (Radio Frequency Identification), and process traceability. The reduction in the cost of microprocessors, along with their decentralization and distribution in highly complex systems have proven to be an extremely powerful and promising architecture. The complexity and mutability of processes create challenges in the areas of control, modeling, and system monitoring. In this context, this work aims to develop and implement the Petri Net Inside RFID Database Management System implemented in ESP32 or PNRD (Petri Net Inside RFID Database) Management System implemented in ESP32 (PELMS), designed to monitor and track RFID components (tags and readers) distributed across an industrial plant, modeled using Petri Nets within the PNRD framework. Using RFID readers installed on production line equipment in a factory, the system will identify the transitions of parts throughout the manufacturing process via tags attached to each part. It will then take actions to ensure that the final product meets the desired quality standards, being capable of automatically detecting any anomalies or non-conformities in the expected process. Both Petri Nets and RFID are essential components for monitoring and managing the production chain, as they enable traceability, identification, and standardization of products. The PELMS plays a crucial role in setting up the PNRD system, as well as monitoring and displaying the status of network elements.

Keywords: RFID, Petri Nets, ESP32, PNRD, PELMS, Embedded Systems, System Control, System Modeling, Solution Modularization.

LISTA DE ILUSTRAÇÕES

Figura 1 - Esquema de funcionamento da tecnologia RFID. Fonte dos ícones: < https://icons8.com/ >, acesso em 07/09/2024.....	20
Figura 2 - Tipos de Etiquetas, Fonte: < https://pt.aliexpress.com/ >, acesso em 15/09/2024.....	20
Figura 3 - Modo RFID Ativo. Fonte: < https://techblogpedia.com/ >, acesso em 15/09/2024.....	21
Figura 4 - Modo RFID Passivo. Fonte: < https://techblogpedia.com/ >, acesso em 15/09/2024.....	21
Figura 5 - Modo RFID Semi-passivo. Fonte: < https://techblogpedia.com/ >, acesso em 15/09/2024.....	22
Figura 6 - Módulos internos do circuito integrado de Tag. Fonte: < https://www.gta.ufrj.br/ >, acesso em 16/09/2024	23
Figura 7 - Diferentes tipos de Etiquetas (TAG). Fonte: < https://www.indiamart.com/ >, acesso em 10/09/2024.....	23
Figura 8 - Etiquetas de acordo com as faixas de frequência de operação. Fonte: < https://pt.aliexpress.com/ >, acesso em 15/09/2024	24
Figura 9 - Representação do código EPC. Fonte: < https://www.gta.ufrj.br/ >, acesso em 16/09/2024.....	25
Figura 10 - Antenas Monoestáticas. Fonte: < https://pt.aliexpress.com/ >, acesso em 15/09/2024.....	26
Figura 11 - Radiação Antena Direcional. Fonte: < http://blog.atlasrfidstore.com >, acesso em 10/09/2024.....	26
Figura 12 - Radiação Antena Omnidirecional. Fonte: < http://blog.atlasrfidstore.com >, acesso em 10/09/2024.....	27
Figura 13 - Feixe de antenas direcionais de diferentes ganhos	27
Figura 14 - Polarização linear, circular e elíptica. Fonte: < https://www.researchgate.net/ >, acesso em 20/09/2024.....	28
Figura 15 - Modelo simples de uma Rede de Petri.....	32
Figura 16 - Exemplo de modelagem de um semáforo com um token no estado verde.	33
Figura 17 - Exemplo de tratamento de concorrência com Redes de Petri.....	34

Figura 18 - Esquema de teste de peça usinada. Fonte: Improving Exception Analysis in PNRD systems using Labelled Free Choice Nets	38
Figura 19 - Visão geral da estrutura PNML. Fonte: https://www.researchgate.net/figure/Overview-of-the-parts-of-PNML_fig1_220783478 , acessado em 20/09/2024.....	40
Figura 20 - Estrutura do módulo Metamodelo da PNML. Fonte: https://www.researchgate.net/figure/The-PNML-meta-model_fig2_220783478 , acessado em 20/09/2024.....	40
Figura 21 - PNRD integrando Redes de Petri e RFID.....	41
Figura 22 - Associação dos termos da equação do vetor de marcação com os elementos armazenados na PNRD (a) e iPNRD (b). Fonte: FONSECA et al., 2018	42
Figura 23 - Exemplo de uma PNRD para verificação de mancais. Fonte: SILVA (2017)	44
Figura 24 - Modelos de Microcontroladores ESP. Fonte: https://www.vidadesilicio.com.br , acessado em 22/09/2024	46
Figura 25 - Mapa de Pinos ESP-WROOM-32 DEVKIT V1. Fonte: https://mischianti.org , acessado em 21/09/2024.....	47
Figura 26 - Modelo ESP-IDF. Fonte: https://docs.espressif.com , acessado em 22/09/2024.....	47
Figura 27 - Raspberry Pi 4 Model B. Fonte: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/ , acessado em 22/09/2024 ...	50
Figura 28 - Modelo OSI de alguns protocolos de comunicação IoT. Fonte: https://www.researchgate.net , acessado em 27/09/2024	53
Figura 29 - Diagrama representativo da arquitetura MQTT. Fonte: https://alfacomp.net/ , acessado em 20/10/2024.....	53
Figura 30 - Infraestrutura de Comunicação do MQTT com o Broker, Publisher e Subscriber. Fonte: https://alfacomp.net/ , acessado em 20/10/2024.....	54
Figura 31 - Software de geração da imagem dos sistemas operacionais disponíveis para instalação no Raspberry Pi. (Acessado em 07/12/2024: https://www.raspberrypi.com/documentation/computers/getting-started.html).....	58
Figura 32 - Teste de Subscrição e Publicação no Broker.	59
Figura 33 - Publicações dos Tópicos do ESP32® Disparador.	60
Figura 34 - Publicações dos Tópicos do ESP32 Recorder.	61

Figura 35 - Subscrição do Runtime do ESP32® Disparador.	61
Figura 36 - Modelo de Rede de Petri de uma planta de teste de qualidade e retrabalho de mancais em um ambiente fabril.	62
Figura 37 - Modelo de Rede de Petri de um mapa das posições que um robô pode ocupar em uma sala.	63
Figura 38 - Arquivos gerados pelo middleware PELMS-MQTT.....	64
Figura 39 – Esquema de funcionamento do middleware PELMS-MQTT.....	65
Figura 40 - Menu e aba "SETUP" do middleware PELMS-MQTT.....	66
Figura 41 – Aba “RUNTIME” do middleware.	67
Figura 42 - Esquema de Ligação do ESP32.	68
Figura 43 - Montagem do ESP32 Recorder e Reader.	68
Figura 44 - Menu de opções no software PELMS-MQTT.....	70
Figura 45 - Janela de abertura de arquivo PNML no software PELMS-MQTT	71
Figura 46 - Popup gerado com a informação de arquivos gerados.	71
Figura 47 - Janela de abertura de arquivo PELMS no software PELMS-MQTT.	72
Figura 48 - Rede de Petri e dados dos reader gerados no middleware PELMS-MQTT.	72
Figura 49 - Aba RUNTIME no middleware PELMS-MQTT.....	73
Figura 50 - Recebimento de dados via MQTT enviadas pelo middleware PELMS-MQTT para o ESP32-Recorder.	73
Figura 51 - Salvamento dos dados da PNRD na Tag no Serial Monitor para o ESP32-Recorder.....	74
Figura 52 - Salvamento dos dados da PNRD na Tag apresentado na tela LCD 16x2..	74
Figura 53 - Recebimento de dados via MQTT enviadas pelo middleware PELMS-MQTT para o ESP32-Reader.	75
Figura 54 - Disparo e salvamento dos dados da PNRD na Tag no Serial Monitor para o ESP32-Reader.....	75
Figura 55 – Disparo e salvamento dos dados da PNRD na Tag apresentados na tela LCD 16x2.	76
Figura 56 - Envio dos dados via MQTT do ESP32-Reader para o middleware PELMS-MQTT referente ao disparo.	76
Figura 57 – Disparo de uma transição no middleware PELMS-MQTT.....	77
Figura 58 – Excessão de uma Tag no middleware PELMS-MQTT.	77

Figura 59 - Rede de Petri e dados dos reader gerados no middleware PELMS-MQTT.	78
Figura 60 - Aba RUNTIME no middleware PELMS-MQTT.	78
Figura 61 - Recebimento de dados via MQTT enviadas pelo middleware PELMS-MQTT para o ESP32-Recorder.	79
Figura 62 - Salvamento dos dados da iPNRD na Tag no Serial Monitor para o ESP32-Recorder.....	79
Figura 63 - Salvamento dos dados da iPNRD na Tag apresentado na tela LCD 16x2.80	
Figura 64 - Recebimento de dados via MQTT enviadas pelo middleware PELMS-MQTT para o ESP32-Reader.	80
Figura 65 – Disparo e salvamento dos dados da iPNRD na Tag no Serial Monitor para o ESP32-Reader.....	81
Figura 66 – Disparo e salvamento dos dados da iPNRD na Tag apresentado na tela LCD 16x2.	81
Figura 67 - Envio dos dados via MQTT do ESP32-Reader para o middleware PELMS-MQTT referente ao disparo.	82
Figura 68 – Disparo de uma transição no middleware PELMS-MQTT.....	82
Figura 69 – Excessão de uma Tag no middleware PELMS-MQTT.	83
Figura 70 - Setup da porta e tipo de acesso no Broker.	II
Figura 71 - Setup do brige entre dois Broker.	III

LISTA DE TABELAS

Tabela 1 - Comparação entre os diversos modos de comunicação RFID	22
Tabela 2 - Classes de Etiquetas. Fonte: (ZELBST; SOWER, 2021, RFID for the Supply Chain and Operations Professional)	24
Tabela 3 - Tipos de etiquetas x frequência. Fonte: (ZELBST; SOWER, 2021, RFID for the Supply Chain and Operations Professional)	24

LISTA DE ABREVIATURAS E SIGLAS

- IoT – Internet das Coisas (em inglês *Internet of Things*)
- IIoT – Internet das Coisas Industrial (em inglês *Industrial Internet of Things*)
- ML – Aprendizado de Máquina (em inglês *Machine Learning*)
- AI – Inteligência Artificial (em inglês *Artificial Intelligence*)
- RFID – Identificação por Radiofrequência (em inglês *Radio Frequency Identification*)
- PN ou RP – Rede de Petri (em inglês *Petri Net*)
- PNRD – Rede de Petri Inserida em Base de Dados RFID (em inglês *Petri Net Inside in RFID Database*)
- iPNRD – PNRD invertida (em inglês *inverted PNRD*)
- LED – Diodo Emissor de Luz (em inglês *Light-Emitting Diode*)
- SMD – Dispositivos Montados em Superfícies (em inglês *Surface Mounted Device*)
- ESP32 – Plataforma eletrônica multipropósito desenvolvido pela Espressif Systems
- PALMS – Sistema de Gerenciamento da Biblioteca PNRD/iPNRD para Arduino (em inglês *PNRD/iPNRD Arduino Library Management System*)
- PNML – Linguagem de Marcação de Rede de Petri (em inglês *Petri Net Markup Language*)
- FTP – Protocolo de Transferência de Arquivos (em inglês *File Transfer Protocol*)
- WoPeD – Designer de Rede de Petri de Fluxo de Trabalho (em inglês *Workflow Petri Net Designer*)
- MQTT – Transporte de Telemetria de Enfileiramento de Mensagens (em inglês *Message Queuing Telemetry Transport*)
- CPU – Unidade Central de Processamento (em inglês *Central Processing Unit*)
- PELMS – Sistema de Gerenciamento da Rede de Petri Inserida em RFID implementada em ESP32 (em inglês *Petri Net Inside RFID Database Management System implemented in ESP32*)
- RADAR – Detecção e Distanciometria por Rádio (em inglês *Radio Detection And Ranging*)
- FDX – Duplex completo (em inglês *Full Duplex*)
- HDX – Meio Duplex (em inglês *Half Duplex*)
- RF – Radiofrequência (em inglês *Radio Frequency*)
- LF – Baixa frequência (em inglês *Low Frequency*)

HF – Alta frequência (em inglês *High Frequency*)

UHF – Ultra alta frequência (em inglês: *Ultra High Frequency*)

SHF – Super Alta Frequência (em inglês *Super High Frequency*)

ISO – Organização Internacional para Padronização (em inglês *International Organization for Standardization*)

EPC – Código Eletrônico do Produto (em inglês *Electronic Product Code*)

TID – Identificador do Fabricante da Etiqueta (em inglês *Tag ID Manufacturer*)

RS-232 – Padrão de comunicação serial, Recomendação Padrão 232 (em inglês *Recommended Standard 232*)

RJ45 – Tomada 45 Registrada (em inglês *Registered Jack-45*)

USB – Barramento Serial Universal (em inglês *Universal Serial Bus*);

RS-485 – Padrão de comunicação serial, Recomendação Padrão 485 (em inglês *Recommended Standard 485*)

I2C - Circuito Inter Integrado (em inglês *Inter-Integrated Circuit*)

TTL – Lógica Transistor-Transistor (em inglês *Transistor-Transistor Logic*)

SPI – Interface Serial Periférica (em inglês *Serial Peripheral Interface*)

NFC – Comunicação de Campo Próximo (em inglês *Near Field Communication*)

XML – Linguagem de Marcação Extensível (em inglês *Extensible Markup Language*)

UML – Linguagem de Modelagem Unificada (em inglês *Unified Modeling Language*)

NDEF – Formato de troca de informações em comunicações por campo de proximidade (em inglês *NFC Exchange Format*)

PCB – Placa de Circuito Impresso (em inglês *Printed Circuit Board*)

UART – Transmissor/Receptor Assíncrono Universal (em inglês *Universal Async Receiver/Transmitter*)

CI – Circuito Integrado (em inglês *Integrated Circuit*)

SoC – Sistema em Chip (em inglês *System on Chip*)

GCC – Coleção de Compiladores de GNU (em inglês *GNU Compiler Collection*)

GPIO – Entrada/Saída de Propósito Geral (em inglês *General Purpose Input/Output*)

RAM – Memória de Acesso Aleatório (em inglês *Random Access Memory*)

LXDE – Ambiente de Desktop Leve X11 (em inglês *Lightweight X11 Desktop Environment*)

GUI – Interface Gráfica de Usuário (em inglês *Graphical User Interface*)

IBM – Máquinas Internacional de Negócio (em inglês *International Business Machines*)

OASIS – Organização para Avanço de Padrões em Informação Estruturada (em inglês *Organization for the Advancement of Structured Information Standards*)

M2M – Máquina para Máquina (em inglês *Machine to Machine*)

TCP – Protocolo de Controle de Transmissão (em inglês *Transmission Control Protocol*)

IP – Protocolo de Internet (em inglês *Internet Protocol*)

OSI – Interconexão de Sistema Aberto (em inglês *Open System Interconnection*)

HTTP – Protocolo de Transferência de Hipertexto (em inglês *Hypertext Transfer Protocol*)

QoS – Qualidade de Serviço (em inglês *Quality of Service*)

TLS – Segurança de Camada de Proteção (em inglês *Transfer Layer Security*)

SSL – Segurança de Camada de Soquete (em inglês *Secure Socket Layer*)

WLAN – Rede Local Sem Fio (em inglês *Wireless Local Area Network*)

Cartão SD – Cartão Digital de Segurança (em inglês *Secure Digital Card*)

MAPL – Laboratório de Planejamento de Automação e Manufatura (em inglês *Manufacturing Automated Planning Lab*)

WSL2 – Subsistema Windows para Linux 2 (em inglês *Windows Subsystem for Linux 2*)

WAN – Rede de Longa Distância (em inglês *Wide Area Network*)

CLP ou PLC – Controlador Lógico Programável (em inglês *Programmable Logic Controller*)

SO ou OS – Sistema Operacional (em inglês *Operating System*);

UFU – Universidade Federal de Uberlândia.

SUMÁRIO

Sumário

1	INTRODUÇÃO.....	12
1.1	Objetivos.....	16
1.2	Justificativa.....	16
1.3	Estrutura do Trabalho	17
2	FUNDAMENTAÇÃO TEÓRICA	18
2.1	RFID	18
2.2	Etiqueta RFID.....	22
2.3	Antena RFID	26
2.4	Leitor RFID	29
2.5	Infraestrutura de Comunicação.....	29
2.6	Aplicação	29
2.7	NFC - <i>Near-Field Communication</i> (Comunicação de Campo Próximo).....	30
2.8	Rede Petri	31
2.8.1	Representação formal da Rede de Petri.....	35
2.8.2	Representação gráfica de um exemplo de Rede de Petri.....	37
2.8.3	PNML	39
2.8.4	PNRD (Rede de Petri Inserida em Base de Dados RFID) e iPNRD (PNRD invertida)	41
2.9	ESP32	45
2.10	Raspberry PI	48
2.11	MQTT	51
3	METODOLOGIA E DESENVOLVIMENTO.....	57
3.1	Estudos Iniciais.....	57
3.2	Instalação do Sistema Operacional no Raspberry PI 4.....	57

3.3	Instalação e Configuração do Broker	58
3.4	Comunicação Publicador-Subscritor	59
3.5	Geração dos arquivos PNML via WoPeD	62
3.6	PELMS-MQTT.....	64
3.6.1	Implementações/Melhorias PELMS-MQTT	65
3.7	Montagem das placas ESP32® Recorder e Reader	67
4	RESULTADOS	69
4.1	Módulo <i>SETUP</i>	69
4.2	Módulo <i>RUNTIME</i>	69
5	CONCLUSÃO.....	84
5.1	Trabalhos Futuros	85
5.1.1	Hardware	85
5.1.2	<i>Back-end e Front-end</i>	85
	REFERÊNCIAS	86
APÊNDICE A	INSTALAÇÃO DO BROKER MOSQUITTO MQTT NA Distro LINUX	I
APÊNDICE B	SETUP DO BROKER MOSQUITTO MQTT NA Distro LINUX	II
APÊNDICE C	MAINWINDOW.PY	IV
APÊNDICE D	PELMSGUI.PY	XXI
APÊNDICE E	TEMPLATE.PY	XXXV
APÊNDICE F	TEMPLATE_BEGIN.INO	XXXIX
APÊNDICE G	TEMPLATE_READER_IPNRD.INO	XLI
APÊNDICE H	TEMPLATE_READER_PNRD.INO	XLVI
APÊNDICE I	TEMPLATE_RECORDER_IPNRD.INO.....	L
APÊNDICE J	TEMPLATE_RECORDER_PNRD.INO	LV
APÊNDICE K	TEMPLATE_WIFIPASSWORD.H	LX

1 INTRODUÇÃO

A indústria está passando por um processo de transformação, denominado Indústria 4.0, uma nova revolução das técnicas industriais, que alguns chamam da quarta revolução industrial, é um movimento global que busca tornar a produção mais eficiente, ágil e inteligente, ela se caracteriza pela automação industrial e pelo uso de tecnologias digitais, tais como a Comunicação entre Dispositivos, Conectividade em nuvem, Rastreabilidade dos Processos, Internet das Coisas (IoT - *Internet of Things*) ou Internet das Coisas Industrial (IIoT), ML - *Machine Learning* (aprendizado de máquina), Robótica avançada, Big data (Grande volume de dados), Inteligência Artificial (AI- *Artificial Intelligence*), e as diferentes técnicas de análise de dados.

A Indústria 4.0 surgiu em um ambiente de grandes mudanças e avanços no mundo industrial, caracterizado pelo aumento na capacidade dos computadores, memória, processamento, armazenamento e velocidades das placas mãe, imensa quantidade de informação digitalizada, novas estratégias de inovação, e redução do custo dos microprocessadores e a sua descentralização e dispersão de sistemas de grande complexidade, tem se mostrado como uma arquitetura extremamente poderosa e promissora. A complexidade e mutabilidade dos processos, geram desafios na área de controle, modelagem e monitoramento de sistemas.

A implementação de sistemas de controle, modelagem e monitoramento na indústria 4.0, pode trazer uma série de benefícios para as empresas, e a utilização da solução de Identificação por Radiofrequência, (RFID - Radio Frequency Identification) é uma abordagem que otimiza a gestão e automação de processos, proporcionando maior agilidade, visibilidade, precisão e eficiência operacional, para que as empresas estejam preparadas para responder rapidamente às demandas do mercado e aos desafios de produção. O RFID, ao capturar dados em tempo real sobre itens, produtos e processos, pode transformar a maneira como as empresas e fábricas operam.

A Identificação por Radiofrequência (RFID - Radio Frequency Identification) é uma tecnologia que usa ondas de rádio eletromagnéticas para identificar, rastrear e registrar dados de objetos, pessoas ou animais, através da leitura de dados em etiquetas, por meio de uma antena de rádio.

A solução de RFID, é composta pelos seguintes elementos:

- O leitor RFID: responsável por criar uma onda eletromagnética (de radiofrequência), que irá receber e decodificar o sinal, bem como pode enviar os dados do sistema de processamento para as etiquetas RFID;

- A etiqueta (Tag RFID): responsável por armazenar os dados recebidos do sistema e podem ser portadas, integradas ou anexada a produtos, materiais, equipamentos, pessoas ou animais;
- A antena RFID: serve para recepção e transmissão das informações em ambos os sentidos; e
- O Sistema de Processamento de Dados: responsável por coletar os dados da solução, processar as informações e realizar ações conforme as necessidades do sistema, bem como pode enviar dados e configurações para os elementos de rede.

A tecnologia RFID possui diversas aplicações, tais como, controle das etapas de produção, distribuição e armazenamento de materiais, peças e produtos, monitoramento e rastreamento de cargas em tempo real, monitoramento de estoques de forma automática, etiquetagem de bagagens, cobrança automática em pedágios e estacionamentos, controle de acesso em hotéis e empresas, localização e rastreamento de pessoas em parques nacionais, detecção de furtos e controle de estoques, gerenciamento de exames e informações de pacientes, gerenciamento de amostras, catalogação e controle de acervo bibliotecário, e outros. A maioria dessas aplicações substitui o tradicional código de barras pelas etiquetas RFID.

A Rede de Petri (PN – Petri Net), denominada como rede de transações, é uma técnica de modelagem matemática usada para representar sistemas, especialmente os sistemas concorrentes. De acordo com Murata (1989), as redes de Petri são consideradas como uma ferramenta gráfica e matemática de representação formal que permite a modelagem, análise e controle de sistemas a eventos discretos que comportam atividades paralelas, concorrentes e assíncronas.

A Rede de Petri permite descrever casos de simultaneidade, paralelismo, assincronismo, conflitos e deadlocks, tornando-a útil para sistemas com fluxos de trabalho complexos, como processos de produção modernos.

Tavares e Saraiva propuseram um método para a integração de Redes de Petri com sistemas RFID, criando PNRD (*Petri Net Inside in RFID Database* – Rede de Petri Inserida em Base de Dados RFID), uma estrutura que permite que as etiquetas RFID contenham informações em tempo real sobre o estado do produto e o processo em que ele está inserido. Isso permite a identificação automática de eventos inesperados na linha de produção. Tavares e Saraiva (2010).

A implementação de uma solução PNRD ou iPNRD ou PNRD invertida (Fonseca et al, 2018) requer desenvolvimento de software, configuração de uma plataforma eletrônica e programação de leitores RFID. Da Silva, Tavares e Ferreira (2018) implementou a biblioteca PNRD e a iPNRD utilizando a plataforma Arduino®.

O Arduino® (ARDUINO, 2024) é uma plataforma eletrônica caracterizada como um sistema embarcado multipropósito compatível com diversos módulos e hardwares, como sensores (leitor RFID, presença, umidade, etc.), atuadores (LEDs, motores, cilindros, etc.) e placas de comunicação (Xbee, Ethernet, WiFi, etc.). É construído em torno de um microcontrolador ATmega, disponível nos formatos SMD ou PID, cujos programas são carregados exclusivamente por meio de uma porta serial.

No entanto a plataforma Arduino®, possui restrições de hardware, software e interfaces de comunicação, além de uma estrutura interna de matrizes e vetores fixas e limitadas, sendo necessário desenvolver e adaptar a biblioteca Arduino para uma nova plataforma eletrônica, a plataforma ESP32®.

O ESP32® (ESP32, 2024) é uma plataforma eletrônica com características de um sistema embarcado multipropósito e IoT, também compatível com vários módulos e hardware, como sensores, atuadores e placas de comunicação. Esta plataforma suporta carregamento de programas via porta serial ou WiFi. Algumas vantagens de usar essas plataformas incluem uma grande comunidade de desenvolvedores e colaboradores, facilidade de desenvolvimento de software, escalabilidade e aplicabilidade em redes coletivas de dispositivos conectados (IoT), baixo consumo de energia, alto desempenho, maior espaço de RAM, processamento de 32 bits e Bluetooth e WiFi integrados, disponibilidade de bibliotecas e suporte para programação orientada a objetos.

A biblioteca ESP32® é útil para ler e escrever tags RFID, disparar transições PNRD/iPNRD de objetos no momento da captura de dados, atualizar o estado do objeto e verificar automaticamente exceções de processo. Os dispositivos ESP32® integrados com leitores RFID devem ser programados com as informações PNRD/iPNRD para interagir com as tags RFID.

Carrijo (2021) desenvolveu o PALMS ou *PNRD/iPNRD Arduino Library Management System* (Sistema de Gerenciamento da Biblioteca PNRD/iPNRD em Arduino). O PALMS é um middleware capaz de integrar ferramentas de modelagem de rede de Petri (através do uso de arquivo exportado PNML – *Petri Net Markup Language* ou Linguagem de Marcação de Rede

de Petri) com PNRD/iPNRD. O PALMS original (Carrijo, 2021) permite a integração de diversos microcontroladores através de conexões FTP – *File Transfer Protocol* ou Protocolo de Transferência de Arquivos, protocolo de comunicação máquina para máquina. O PALMS interpreta arquivos PNML (arquivos gerados em ferramentas de modelagem de Petri Net, ex. WoPeD), e os converte para arquivos “.palms”, para que estes sejam transferidos para microcontroladores Arduino®, equipado com leitor RFID, módulo WIFI ou Ethernet e cartão de memória e já configurados com biblioteca PNRD/iPNRD.

O PALMS original possui várias limitações, a saber, necessita de *Shield* ethernet no Arduino, o Arduino possui limitações de hardware (memória, velocidade de processamento, interfaces, conversão analógico-digital, armazenamento, etc.), consequentemente o PALMS original não permite a implementação de modelos complexos de Petri Net, possui pouca eficiência, sua estrutura interna de matrizes e vetores tem que ser fixa, não permite trabalhar com ponteiros, não possui o sistema *whatchdog* (temporizador que reseta o sistema se houver algum travamento) dentre outras limitações.

Para o funcionamento PALMS com conexão MQTT, é necessário a utilização de um *Broker*, ou seja, um servidor para comunicação MQTT. Ele pode ser tanto local, instalado localmente, ou por meio de algum servidor online disponível para acesso. As vantagens de implementação local permitem um tempo de resposta rápido e garante uma segurança nos dados que são armazenados no *Broker*. Os *Brokers* online que fornecem esses benefícios demandam um pagamento para a utilização e a manutenção.

O protocolo MQTT por ter características de um protocolo simples e leve, não demanda de um servidor potente, ou seja, servidores pequenos ou locais são capazes de suportar a taxa de transmissão entre cliente e servidor. Sendo assim, computadores e minicomputadores são uma escolha ideal para utilização como *Broker*, pois possuem CPUs capazes de atender a taxa de comunicação solicitada pelo Broker e não geram muitos gastos para o consumidor. O Raspberry Pi 4 é um exemplo de minicomputador que atende esses requisitos. Ele é uma plataforma utilizada em jogos leves, controle de casas inteligentes, IIoT.

No intuito de resolver estes problemas e limitações, foi proposto a implementação do middleware PALMS em uma nova plataforma eletrônica mais robusta e completa, denominada PELMS - *PNRD/iPNRD ESP32 Library Management System* (Sistema de Gerenciamento da Rede de Petri Inserida em Base de Dados RFID implementada em ESP32).

1.1 Objetivos

O principal objetivo deste projeto consiste na adaptação da biblioteca *PNRD Arduino Library* e do *middleware* PALMS para a plataforma ESP32. Os objetivos específicos desse projeto são:

1. Otimizar os métodos de manipulação da PNRD e de gravação em etiquetas;
2. Instalação do sistema operacional Raspbian no *Raspberry Pi 4 Model B*;
3. Instalação e configuração do Broker no Raspberry Pi;
4. Melhorias do *front-end* do *middleware*:
 - a. Cor das janelas, abas e botões;
 - b. Inserção do título da janela;
5. Implementação dos *template* que serão gerados pelo *middleware*;
6. Implementação do funcionamento em tempo real do módulo RUNTIME no *middleware*;
7. Geração de dois modelos de Rede de Petri na ferramenta WoPeD para os testes do *middleware*.

1.2 Justificativa

Este trabalho apresenta um modelo de geração e manipulação de Redes de Petri para funcionamento em *Cloud* (Nuvem), desenvolvido para uma plataforma ESP32, e visa melhorar as limitações do *middleware* PALMS, ampliando a capacidade de processamento e interoperabilidade, a capacidade de gravação e armazenamento de dados, a escalabilidade e aplicabilidade em IoT, e disponibilidade para operar em múltiplos sistemas operacionais.

Alguma das disciplinas que auxiliaram neste trabalho foram, Algoritmos e Programação de Computadores (Programação em linguagem C e C++), Arquitetura e Organização de Computadores (Aplicabilidade de Programação em nível de Hardware), Sistemas Operacionais (Conceitos sobre Sistemas Operacionais e sua aplicação), Instrumentação (Aprendizado dos elementos que compõe os *hardware* do sistema PALMS), Arquitetura de Redes de Computadores (Redes e Protocolos de comunicação), Sistemas Digitais para Mecatrônica (Programação em outras linguagens – Python), Redes Industriais (Tipos estrutura de redes, protocolos de comunicação e tempo de comunicação entre dispositivos) e Automação Industrial (Redes de Petri e PNRD).

1.3 Estrutura do Trabalho

O trabalho foi estruturado em fundamentação teórica (Capítulo 2), metodologia e desenvolvimento (Capítulo 3), resultados (Capítulo 4), conclusão (Capítulo 5), referências e apêndices.

Fundamentação Teórica: foram apresentados as tecnologias e conceitos teóricos para o desenvolvimento do trabalho.

Metodologia e Desenvolvimento: foram apresentados a aplicação dos conceitos estudados no capítulo anterior e o desenvolvimento do projeto.

Resultados: foram apresentados os resultados obtidos após a utilização do projeto desenvolvido.

Conclusão: foram apresentados o impacto e a aplicação do trabalho, e como foram atendidos os objetivos do projeto, e também foram apresentadas as aplicações e inovações que podem ser implementados no trabalho.

Referências: foram apresentados os títulos e os links que foram utilizados para o desenvolvimento do trabalho.

Apêndice: foram apresentados informações e passos utilizados no desenvolvimento deste projeto.

2 FUNDAMENTAÇÃO TEÓRICA

A solução desenvolvida nesse trabalho tem um aspecto amplo e interdisciplinar, com possibilidade de implementação em diversos tipos de sistemas que possa trabalhar com Redes de Petri. Este capítulo apresenta a fundamentação teórica dos elementos e conteúdo estudados para a confecção e o embasamento técnico do projeto.

2.1 RFID

A tecnologia RFID (*Radio Frequency Identification*) ou Identificação por Radio Frequência remonta aos tempos da Segunda Guerra Mundial, onde as potências militares envolvidas na guerra, estavam utilizando RADAR (*Radio Detection And Ranging* - Detecção e Distanciometria por Rádio), aparelho desenvolvido por Robert Alexander Watson-Watt em 1934, com a finalidade de detectar objetos a longa distância, emitindo e recebendo ondas eletromagnéticas (HWANG, 2013).

O aparelho detectava aviões que se aproximavam a quilômetros de distância, no entanto não permitia identificar quais aviões pertenciam aos inimigos e quais pertenciam aos aliados, assim sob a liderança de Robert Alexander Watson-Watt, foi desenvolvido o primeiro sistema ativo de identificação por radiofrequência, onde transmissores colocados em aviões ao receberem sinais das estações de radar no solo, começavam a transmitir um sinal de volta que identificava que a aeronave era amiga.

O funcionamento do RFID consistia basicamente em enviar um sinal para um transponder, que acordava e refletia o sinal de volta (sistema passivo) ou transmitia um novo sinal (sistema ativo). Nas décadas de 50 e 60, cientistas e acadêmicos dos Estados Unidos, Europa e Japão divulgaram diversas pesquisas a respeito de como a energia de radiofrequência poderia ser utilizada para identificar objetos remotamente.

Desde então, os sistemas RFID evoluíram bastante, e hoje são amplamente utilizados nas indústrias aeroespacial, farmacêutica, manufatura, pecuária, automotiva, logística e outras, os sistemas de rastreamento RFID estão em um nível muito mais complexo, proporcionando, maior precisão e velocidade dos dados, inventário aprimorado, rastreamento de ativos em tempo real, maior segurança e redução de custos e mão de obra (PDA Soluções, 2021).

Nos últimos anos, vários grupos econômico, social, militar, saúde e político passaram a enxergar o potencial inovador da tecnologia para melhorar a eficiência de seus processos

usando a tecnologia RFID. O uso de etiquetas RFID aumentou consideravelmente, e seus desempenhos de comunicação e nível de padronização estão sendo cada vez mais refinados.

O RFID consiste em uma tecnologia que utiliza comunicação via ondas de rádio para trocar dados entre um leitor e uma etiqueta eletrônica atachada a um objeto, principalmente para o propósito de identificação e rastreamento (Coskun, Ok e Ozdenizci, 2011), basicamente a comunicação é baseada em uma troca de ondas eletromagnéticas entre etiquetas (Tag's) e os leitores (*Reader's*). Esta tecnologia de Identificação Automática e Captura de Dados (Auto-ID) fornece uma ampla gama de aplicações para identificar objetos, não permitidas pelo tradicional código de barras e tecnologias anteriores.

O RFID é muito utilizado em diversas áreas do setor de produção e serviços, tais como, distribuição e armazenamento de materiais, peças e produtos, monitoramento e rastreamento de cargas em tempo real, monitoramento de estoques de forma automática, etiquetagem de bagagens, cobrança automática em pedágios e estacionamentos, controle de acesso em hotéis e empresas, localização e rastreamento de pessoas em parques nacionais, detecção de furtos e controle de estoques, gerenciamento de exames e informações de pacientes, gerenciamento de amostras, catalogação e controle de acervo bibliotecário, e outros. Com o RFID, é possível se ter um controle mais ágil, já que não é preciso contato direto entre os funcionários e os produtos, e podendo estes ser localizados e rastreados automaticamente.

Uma vantagem na utilização do RFID para a identificação de objetos é a capacidade que as etiquetas RFID possuem de alterar dinamicamente a sua informação interna. Além disso, se comparada ao código de barras, a tecnologia RFID facilita a leitura, pois o sensor não precisa apontar diretamente para a etiqueta, só precisa estar em seu alcance. Também é possível ler várias etiquetas de uma única vez, aumentando a rapidez do processo.

O sistema RFID é composto por diversos elementos, tais como etiquetas (Tag's), antenas, leitores (*Reader's*), aplicativos (Software), e infraestrutura de comunicação (cabos, hubs, multiplexadores, ...), que permitem uma comunicação sem fio entre os elementos, possibilitando a troca de informações entre os leitores e etiquetas atachadas a objetos, pessoas ou animais. Essa troca de informações ocorre de acordo com um dos dois processos principais de comunicação: *full duplex* (FDX) e *half duplex* (HDX).

No processo HDX ocorre a transferência de dados da etiqueta para o leitor e vice-versa, não simultaneamente, ou seja, a transmissão é realizada nos dois sentidos, mas não ao mesmo tempo, quando um dos dispositivos necessita transmitir, o outro deve “concordar” em receber

e assim transmitir a informação. No processo FDX ocorre a transferência de dados da etiqueta para o leitor ao mesmo tempo em que é feita a transferência de dados do leitor para a etiqueta, ou seja, a transmissão ocorre nos dois sentidos simultaneamente.

A Figura 1 mostra como esses componentes se interagem. Na maioria das aplicações de RFID, o fluxo de dados ocorre da seguinte forma:

- 1) A aplicação realiza uma requisição de leitura;
- 2) Essa requisição é informada ao leitor por meio da infraestrutura de comunicação;
- 3) O leitor gera um sinal de radiofrequência (RF) para detectar etiquetas próximas;
- 4) A antena transmite, via *broadcast*, o sinal do leitor;
- 5) Ao receber o sinal, a etiqueta envia uma resposta à antena através da modulação do sinal recebido;
- 6) O leitor recebe o sinal RF modulado da antena e decodifica a informação presente.
- 7) Utilizando a infraestrutura de comunicação, o leitor informa os dados recebidos para a aplicação.

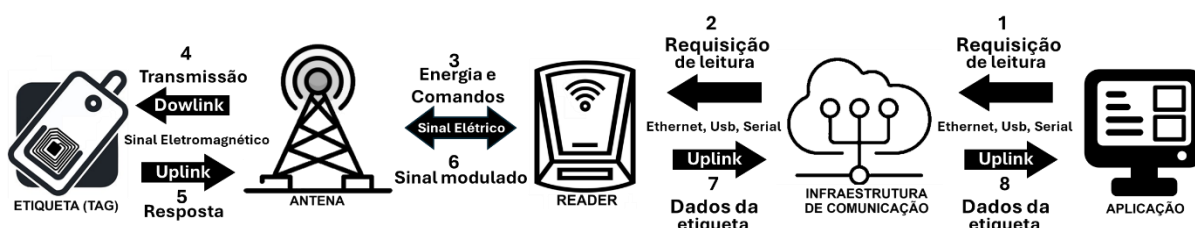


Figura 1 - Esquema de funcionamento da tecnologia RFID. Fonte dos ícones: <<https://icons8.com/>>, acesso em 07/09/2024

O RFID possui três modos de funcionamento, ativo, passivo e semi-passivo (Figura 2), classificados de acordo com a forma que a etiqueta ou tag se comunica com o Leitor ou *reader* (MUROFUSHI, 2011).

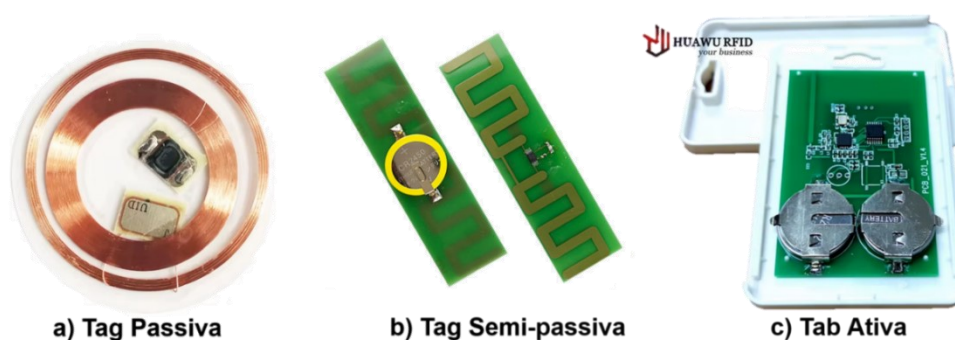


Figura 2 - Tipos de Etiquetas, Fonte: <<https://pt.aliexpress.com/>>, acesso em 15/09/2024

O modo RFID ativo utiliza etiquetas (Tag's) com fonte de energia própria (Figura 3), normalmente uma bateria, e transmitem sinais periodicamente sem quaisquer problemas. As

etiquetas ativas são frequentemente usadas para rastreamento e localização de cargas valiosas e têm um alcance de leitura maior, permitindo rastreamento em tempo real e utilização de Leitores (*Reader's*) de baixa potência. Um exemplo do seu uso, são em pedágios e ambiente de alta velocidade, quando as etiquetas estão a caminho do leitor (Reader), elas transmitem o sinal a uma distância de leitura maior.



Figura 3 - Modo RFID Ativo. Fonte: < <https://techblogpedia.com/>>, acesso em 15/09/2024

No modo RFID passivo (Figura 4), as etiquetas (Tag's) não possuem fontes de energia própria, elas recebem energia das ondas eletromagnéticas enviadas pelo leitor, que induzem uma corrente na antena da etiqueta para alimentar o sensor e executar o circuito do chipset. As etiquetas passivas ficam adormecidas até receberem um sinal do leitor e, então, refletem um sinal de volta para o leitor com as informações de sua identificação. Elas são mais baratas e são amplamente usadas para etiquetas de produtos e paletes.



Figura 4 - Modo RFID Passivo. Fonte: < <https://techblogpedia.com/>>, acesso em 15/09/2024

O modo RFID semipassivo (Figura 5) utiliza etiquetas (Tag's) semipassiva semelhantes às etiquetas ativas, pois possuem uma bateria interna que executa o circuito da etiqueta, não para gerar e transmitir periodicamente um sinal, mas para alimentar o sensor, executar o circuito no chip e reforçar o sinal de retorno, a etiqueta se comunica com o leitor (*Reader*) de forma semelhante ao modo RFID passivo, usando retrodispersão para transmitir as informações moduladas quando induzidas pelo leitor. As etiquetas semipassivas têm um alcance de leitura

maior do que as etiquetas passivas, pois possuem fonte de energia interna para amplificar o sinal de retorno.



Figura 5 - Modo RFID Semi-passivo. Fonte: < <https://techblogpedia.com/>>, acesso em 15/09/2024

A Tabela 1 abaixo mostra uma comparação entre os diferentes modos de comunicação RFID.

Tabela 1 - Comparação entre os diversos modos de comunicação RFID

	Ativo	Passivo	Semi-passivo
Fonte de energia	Interna nas etiquetas	Energia transmitida pelo leitor via RF	Energia Interna e energia recebida do leitor via RF
Bateria	Presente	Ausente	Presente
Disponibilidade energética e propagação de sinal	Energia Contínua e Propagação periódica de sinal	Energia apenas no raio de alcance do leitor e refletem o sinal de volta	Energia Contínua e reflete o sinal de volta quando no raio de alcance do leitor
Intensidade do Sinal entre a Etiqueta e Leitor	Alta	Baixa	Média ou Alta
Alcance de comunicação	Longo alcance	Curto alcance	Médio ou longo alcance
Armazenamento de dados	Grande	Baixa	Média ou Grande
Custo	Alto	Baixo	Médio ou alto

2.2 Etiqueta RFID

As Etiquetas (ou Tag's) são transponders de ondas de Rádio Frequência, dispositivos eletrônicos construídos com microchip ou circuito integrados com uma pequena antena e um encapsulamento, podendo ser compostas por memória e processador, controlados por um sistema operacional ou por microchips de memória não-programável. Em sua maioria, seu principal componente é um microchip capaz de armazenar informação, podendo guardar somente seu número de identificação, ou armazenar até 64 KB de informação (FUJITISU LTD., 2014).

O microchip é a parte eletrônica da etiqueta RFID, e é formado por centenas de microcomponentes agrupados microscopicamente (Figura 6). É responsável por decodificar o sinal de entrada, processar as informações e gerar o sinal de saída modulado e encaminhá-lo para a antena.

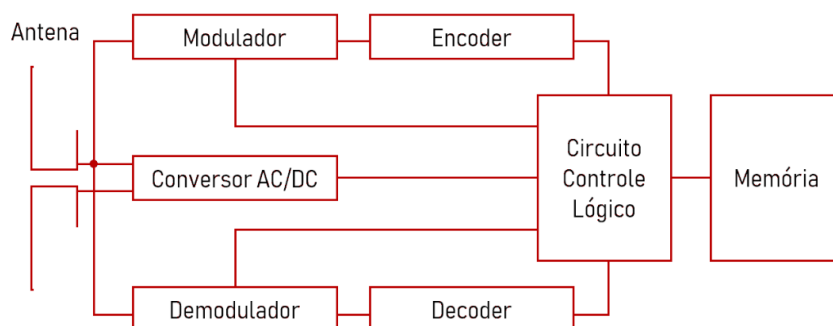


Figura 6 - Módulos internos do circuito integrado de Tag. Fonte: < <https://www.gta.ufrj.br/>>, acesso em 16/09/2024

As etiquetas são comercializadas em vários tipos, tamanhos, resistências e modelos de encapsulamento (Figura 7). Essa grande variedade se dá pela existência de muitos tipos distintos de aplicações e modo de comunicação a ser utilizado para o sistema RFID.



Figura 7 - Diferentes tipos de Etiquetas (TAG). Fonte: < <https://www.indiamart.com/>>, acesso em 10/09/2024

As etiquetas podem ser do tipo *rewrite* (regravável), em que é possível modificar a informação salva por meio de ondas RF, enquanto em etiquetas *read-only* (somente leitura),

essa alteração só é possível reprogramando eletronicamente o microchip da etiqueta (AHSAN, SHAH, KINGTON, 2010). Em alguns casos, as etiquetas podem ser mistas, apresentando áreas de memória do tipo *read-only* e do tipo *rewrite*.

As etiquetas também podem ser definidas por classes (Tabela 2) e por frequências de operação (Tabela 3), com relação a classes, elas indicam a capacidade da etiqueta para certas funções, como de gravação ou escrita, e podem indicar se a etiqueta é do tipo ativa ou passiva. As etiquetas podem ser classificadas quanto a faixa de frequência de operação em Baixa Frequência (LF – *Low Frequency*), Alta Frequência (HF – *High Frequency*) e Ultra Alta frequência (UHF – *Ultra High Frequency*) (Figura 8). A etiqueta UHF passiva mais comumente usada é a de Classe 1 Gen 2. Gen 2 refere-se aos requisitos de conformidade para a faixa de operação UHF de 860 a 960 MHz para etiquetas RFID passivas.



Figura 8 - Etiquetas de acordo com as faixas de frequência de operação. Fonte: < <https://pt.aliexpress.com/>>, acesso em 15/09/2024

Tabela 2 - Classes de Etiquetas. Fonte: (ZELBST; SOWER, 2021, *RFID for the Supply Chain and Operations Professional*)

Classes	Descrição
0	Somente leitura
1	Uma escrita, muitas leituras
2	Regravável
3	Semi-passiva
4	Ativa

Tabela 3 - Tipos de etiquetas x frequência. Fonte: (ZELBST; SOWER, 2021, *RFID for the Supply Chain and Operations Professional*)

Tipo	Descrição	Distância
Baixa frequência (LF)	125 - 134,3 kHz	Curta (até 0,1 Metros)
Alta frequência (HF)	13,56 MHz	Média (0,3 Metros)
Ultra alta frequência (UHF)	860 - 960 MHz	Até 3 Metros

Há diversas padronizações para o RFID, e entre elas se destacam a ISO (*International Organization for Standardization*), Organização Internacional para Padronização e a

EPCglobal, havendo várias normas e definições de topologias, parâmetros de frequências, regras de codificação, protocolos de dados e outros.

As etiquetas possuem um Código Eletrônico do Produto do inglês *Electronic Product Code* (EPC) associado a um registro (alfanumérico) exclusivo, permitindo que um objeto, seu local de origem, sua data de produção e outras informações sejam atreladas a esse registro. (ZELBST; SOWER, 2021)

O EPC visa estabelecer um código unitário atrelado a uma etiqueta para identificar objetos. O código numérico gravado na memória da etiqueta é formado por um número do fabricante, mais a classe do objeto e o número de série. Esse valor numérico, associado a uma base de dados, pode agregar um conjunto ilimitado de informações a respeito da etiqueta e de seu hospedeiro.

Um microchip em uma etiqueta passiva básica pode conter 96 bits de informação (Figura 9). Cada microchip possui quatro bancos de informações: Banco 0 [00] (Reservado para senha de acesso e senha de inabilitação), Banco 1 [01] (usado para armazenar o Código eletrônico do produto - EPC, o Protocolo de controle e o Verificador cíclico de redundância - CRC-16), Banco 2 [10] (usado para armazenar o Identificador do Fabricante da Etiqueta – TID *Manufacturer*) e Banco 3 [11] (usado para armazenar dados dos usuários).

Além de chips com 96 bits muitos fabricantes desenvolveram chips de Tags com memórias EPC de 128, 240 e 496 bits. Alguns chips UHF fornecem memória de até 512 e 640 bits, além da memória EPC para o armazenamento de informações adicionais como data, hora e identificação do produto.

EPC Type	Manufacturer	Product Type	Unique Item
01 . 1234567 . 891011 . 001122DBC			
Header 8-bits	EPC Manager 28-bits	Object Class 24-bits	Serial Number 36-bits
256 Combinations possible	268,435,456 Combinations possible	16,777,216 Combinations possible	68,719,476,736 Combinations possible
(or 7.9×10^{28}) unique identities			

Figura 9 - Representação do código EPC. Fonte: < <https://www.gta.ufrj.br/>>, acesso em 16/09/2024

2.3 Antena RFID

A antena é o componente responsável pela transmissão e recebimento das ondas eletromagnéticas usadas na transferência de informação entre o leitor e as etiquetas. Dependendo das suas características construtivas, ela pode favorecer a leitura de etiquetas em certas direções e distâncias. (ZELBST; SOWER, 2021)

A antena opera como uma agente intermediário entre as etiquetas e o leitor, realizando a modulação dos sinais de radiofrequência enviados pelo leitor, e a demodulação dos sinais recebidos das etiquetas. A Figura 10 apresenta uma antena monoestática e algumas de suas utilizações, é o tipo mais comum, podendo ser utilizada em ambientes internos e externos, e funciona transmitindo e recebendo sinais no mesmo equipamento. Existem antenas que só realizam uma das operações, transmissão ou recepção de sinais, sendo chamadas de biestáticas.

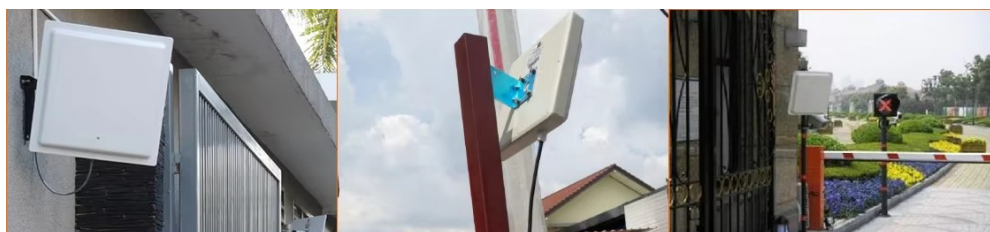


Figura 10 - Antenas Monoestáticas. Fonte: < <https://pt.aliexpress.com/>>, acesso em 15/09/2024

As antenas são classificadas quanto à diretividade, ganho, polarização e frequência de operação. Quanto à diretividade, a antena pode ser direcional, quando ela concentra o sinal em uma direção específica, em formato de cone ou feixe, irradiando para apenas uma direção em que ela esteja apontada (Figura 11), ou omni-direcional, que propaga ondas eletromagnéticas em todas as direções perpendiculares a um eixo, com pouca variação na vertical (Figura 12).

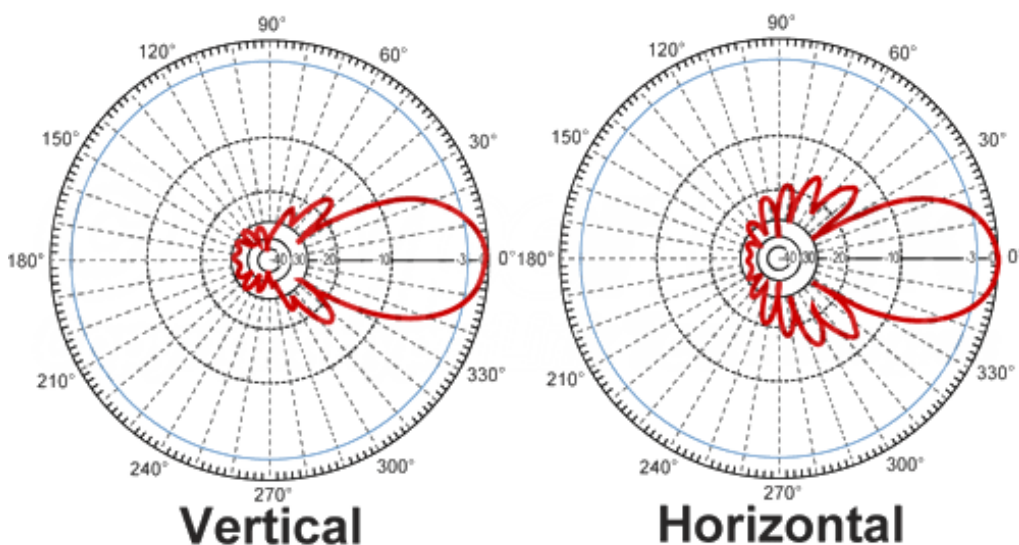


Figura 11 - Radiação Antena Direcional. Fonte: < <http://blog.atlasrfidstore.com>>, acesso em 10/09/2024

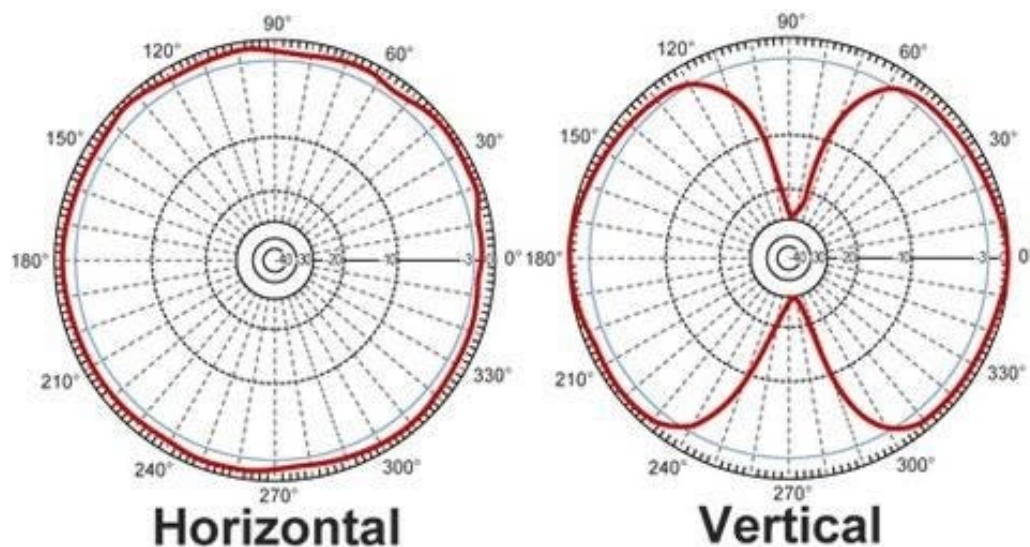


Figura 12 - Radiação Antena Omnidirecional. Fonte: <<http://blog.atlasrfidstore.com>>, acesso em 10/09/2024

Quanto ao ganho a classificação permite determinar quantas vezes a antena ampliará o sinal do leitor, a medida é dada em decibéis (dB) e é atrelada à largura do feixe emitido por uma antena direcional (Figura 13). Sendo que uma antena de menor ganho implica numa maior largura de feixe, e vice-versa.

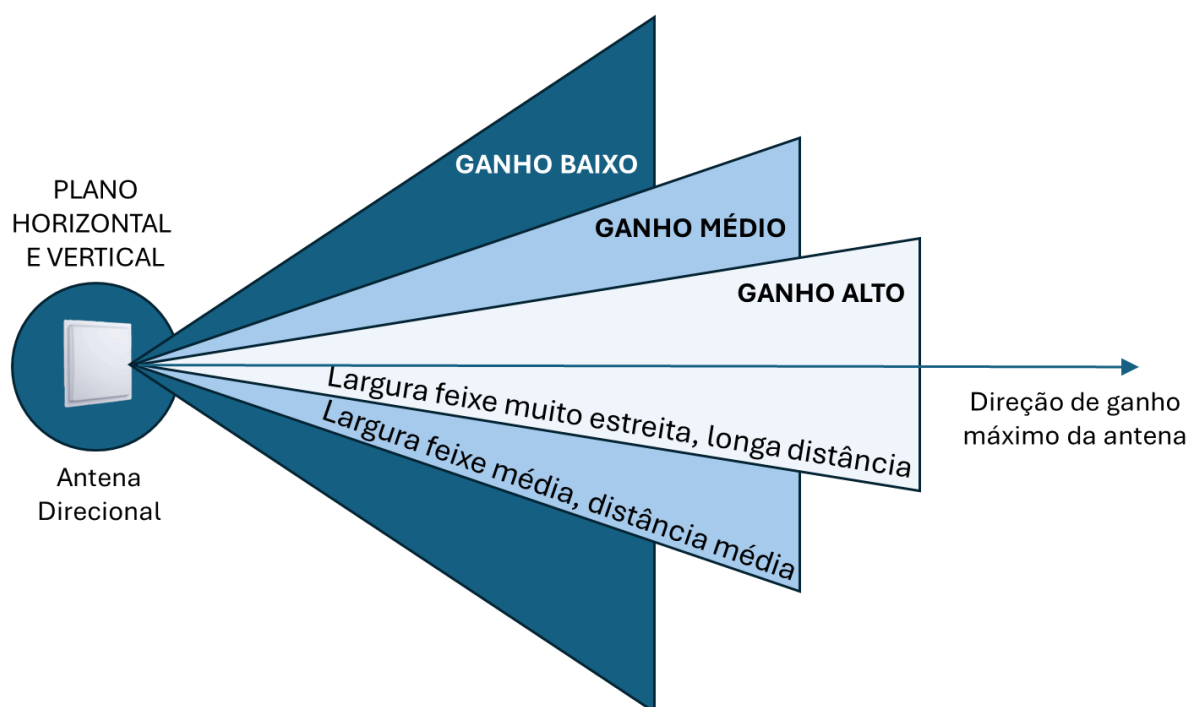


Figura 13 - Feixe de antenas direcionais de diferentes ganhos

Quanto a polarização a classificação permite identificar como o sinal de rádio frequência da antena propaga pelo meio depois de enviado pela antena, podendo ser linear, circular ou elíptica (Figura 14). A polarização da antena está ligada a orientação do vetor elétrico da onda eletromagnética, e é determinada pela estrutura física da antena e pela sua orientação.

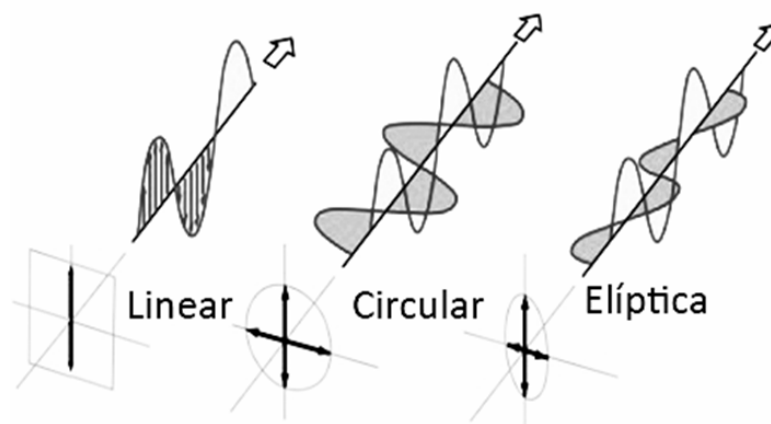


Figura 14 - Polarização linear, circular e elíptica. Fonte: <P <https://www.researchgate.net/> >, acesso em 20/09/2024.

A polarização linear, é caracterizada pela emissão em apenas um plano, normalmente horizontal ou vertical. Este tipo de antena exige que as etiquetas a serem lidas sempre estejam em uma orientação fixa para uma leitura confiável. Como este tipo de antena apresenta irradiação concentrada em apenas um plano, o sinal consegue se propagar por maiores distâncias com maior potência.

Já as antenas de polarização circular ou elíptica, emitem sinais de maneira helicoidal, podendo ser no sentido horário ou anti-horário, transmitindo o sinal em vários planos simultaneamente, a leitura das etiquetas pode ser feita em diversas orientações, porém, a distância de leitura é menor.

Também deve-se levar em consideração a conexão entre a antena e o leitor, realizada por um cabo coaxial que gera também uma perda de potência em dB, em função do comprimento do cabo e seu isolamento.

As antenas são classificadas quanto à frequência em baixa frequência (LF – *Low Frequency*: 125–134 KHz), alta frequência (HF – *High Frequency*: 13.56 MHz), ultra alta frequência (UHF – *Ultra High Frequency*: 860–960 MHz) e Micro-ondas (2.5 GHz e acima). A frequência é um fator que define a velocidade de transferência de dados entre o leitor e as etiquetas RFID. Quanto menor a frequência, mais lenta é a taxa de transmissão.

A distância entre o leitor e a etiqueta RFID varia de acordo com a frequência da antena e depende do tipo de etiqueta utilizado, em regra as antenas podem ter os seguintes alcances:

- LF: Alcance de alguns centímetros até 0,5 metros
- HF: Alcance de até 1 metro
- UHF: Alcance de até 12 metros

- Micro-ondas: Alcance de mais de 10 metros

2.4 Leitor RFID

O leitor é um elemento básico da solução RFID, toda a indução e análise de sinais são feitas pelo leitor RFID, ele é composto por componentes eletrônicos, tais como microchips, microprocessador, transmissor e receptor de rádio frequência, antena, interfaces de entrada e saída, RS-232, RJ45, USB, etc. O leitor é responsável por gerar e interpreta os sinais recebidos pela antena. Ele embute algoritmos contra colisões e pode, por vezes, operar com mais de uma frequência de sinal (AHSAN, SHAH, KINGTON, 2010). Alguns leitores têm a capacidade de conectar-se com múltiplas antenas, enquanto outros estão acoplados a uma única antena, formando um dispositivo único.

O leitor RFID controla diretamente a antena, de maneira a modular os comandos de leitura e escrita a serem enviados para as etiquetas via rádio frequência, e demodular as respostas recebidas pela antena advindas da etiqueta.

A comunicação entre o leitor e a etiqueta é gerenciado e controlado por protocolos RFID, tais como o ISO 15693 e o ISO 18000-3 para HF, o ISO 18000-6 e o EPC 18000-6 para UHF. Basicamente, quando o leitor é ligado, ele começa a emitir um sinal na banda de frequência selecionada (tipicamente 860-915 MHz para UHF ou 13,56 MHz para HF). Qualquer etiqueta correspondente nas proximidades do leitor detectará o sinal e usará a energia dele para acordar e fornecer energia operacional para seus circuitos internos. Uma vez que a etiqueta tenha codificado o sinal como válido, ela responde ao leitor e indica sua presença modulando (afetando) o campo do leitor.

2.5 Infraestrutura de Comunicação

A infraestrutura de comunicação é o canal pelo qual o leitor transmitirá a informação recebida para a aplicação de alto nível. Entre as formas de comunicação possíveis pode-se citar os protocolos seriais como o RS-232, RS-485, I2C TTL e o SPI, protocolos de rede ethernet, USB, ou ainda protocolos sem fio como o WiFi e o Bluetooth.

2.6 Aplicação

A aplicação é o software que vai implementar a solução RFID a ser implementada, ela vai interagir com os elementos do sistema para enviar e receber informações. A aplicação pode estar

ligada a um banco de dados de modo a sincronizar a informação do banco com a leitura das informações da etiqueta e/ou fornecer uma interface para o usuário interpretar os dados lidos.

A maioria das aplicações está ligada ao setor de manufatura, para monitoramento e controle da produção e gestão de cadeias de suprimentos. Outros estudos consideraram o uso de RFID como uma via para simplificar o controle e o gerenciamento de sistemas complexos, como, gerenciamento de processos de reciclagem, rastreabilidade de mercadorias para as indústrias, controle de estoque, gerenciamento de fluxo de pacientes em unidades de saúde, roteamento e manuseio de bagagem em aeroportos, segurança alimentar e rotulagem, rastreamento e gerenciamento de ativos em edifícios, etc.

2.7 NFC - *Near-Field Communication* (Comunicação de Campo Próximo)

NFC é uma tecnologia de comunicação sem fio que usa indução de campo magnético para permitir a comunicação entre dispositivos quando eles são tocados juntos ou colocados a pouca distância um do outro. O NFC pode ser considerado uma extensão do RFID, possuindo o mesmo princípio de funcionamento e operando na faixa de alta frequência (HF) da tecnologia RFID (frequência de 13,56 MHz) (IGOE; COLEMAN; JEPSON, 2014). No entanto, ele pode fazer mais do que apenas ler ids e gravar dados no destino.

O NFC foi criado em 2003 pelas empresas Sony e Philips com o objetivo de ser uma tecnologia de conexão de curta distância de forma segura. Logo após sua criação, a Organização Internacional para padronização adotou a ISO/IEC 18092 como interface e protocolo de fundação do NFC.

Os dispositivos NFC podem operar em modos de leitura/gravação, emulação de cartão e modo ponto a ponto, permitindo uma variedade de aplicações, desde pagamentos móveis até troca de dados entre dispositivos, podendo ser utilizado em diversos dispositivos, como celulares, relógios inteligentes, cartões de crédito e débito, pulseiras e chaveiros, etc.

As diferenças mais relevantes entre RFID e NFC são, a frequência de operação, alcance, modo de comunicação, e etiquetas utilizadas. Os dispositivos NFC possuem dois modos de comunicação que são idênticos ao RFID, com etiquetas passivas e ativas, e mais três modos de operação, com leitores/gravadores para leitura e gravação de dados do destino, emuladores de tag, agindo como etiquetas RFID quando estão no campo de outro dispositivo NFC ou RFID, modo ponto a ponto, no qual as trocam dados ocorrem em ambas as direções, e recursos de segurança, como criptografia de dados, para proteger a comunicação entre dispositivos.

2.8 Rede Petri

A Rede de Petri (RP ou PN – *Petri Net*) é uma técnica formal e abstrata de modelagem de sistemas concorrentes, desenvolvida pelo matemático Carl Adam Petri em 1962, em uma abordagem de representação gráfica e processos matemáticos equivalentes para sistemas dinâmicos de estrutura estática. A Rede de Petri fornece uma representação intuitiva baseada em grafos com semântica formal, composto de lugares (*Places*) e transições (*Transitions*) ligados por arcos (*Arcs*).

Uma rede de Petri é um tipo de grafo bipartido direcionado e preenchido por três tipos de objetos, lugares (representados por círculos ou elipses), que armazenam itens indistinguíveis, denominados fichas ou tokens (representados por pontos) e transições (representados por barras, retângulos ou quadrados), que têm a função de retirar e colocar tokens nos lugares aos quais estão conectados por arcos (representado por linhas ou arcos) seguindo uma regra de disparo (*Fire*).

Em sua forma mais simples, uma rede de Petri pode ser representada por uma transição juntamente com um lugar de entrada e um lugar de saída. Esta representação elementar pode ser usada para vários aspectos de sistemas modelados. Por exemplo, uma transição e seu lugar de entrada e lugar de saída podem ser usados para representar um evento de processamento de dados, com seus dados de entrada e saída em um sistema de processamento de dados.

A representação gráfica e matemática permite análises completas de sistemas (como análise de espaço de estado, gargalos de desempenho e prevenção de inter-bloqueio (*deadlock*) (SILVA, 2017).

A estrutura estática do sistema é representada pelos grafos na Rede de Petri enquanto o dinamismo é representado pela alocação de tokens através do Rede de Petri, dessa forma, a distribuição dos tokens entre os lugares podem mudar de acordo com as regras definidas na Rede de Petri, enquanto a estrutura do sistema permanece fixa.

Em uma Rede de Petri, os lugares são ligados a transições por meio de arcos direcionados. Um arco obrigatoriamente liga um lugar a uma transição, não podendo haver a ligação direta entre duas transições ou dois lugares. A Figura 15 representa um modelo simples de uma Rede de Petri com 4 transições e 5 lugares.

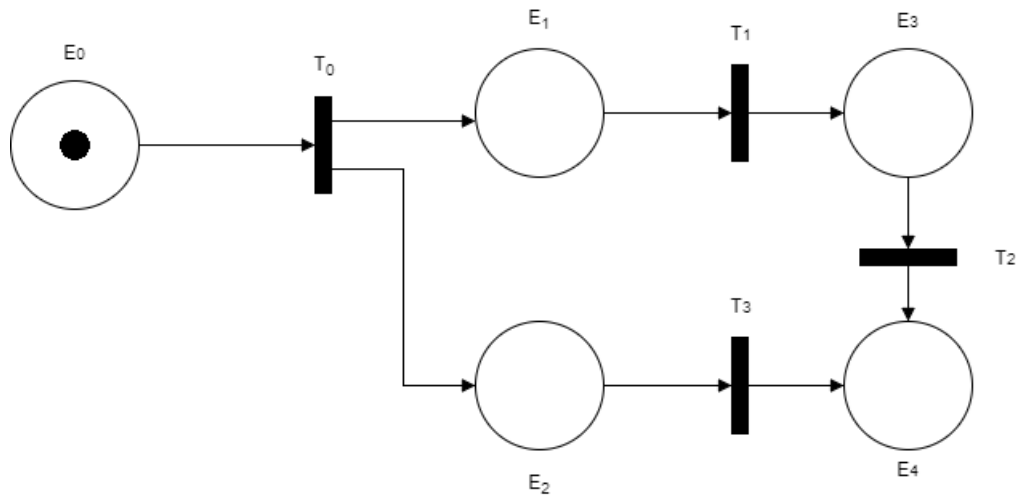


Figura 15 - Modelo simples de uma Rede de Petri

Os lugares dos quais um arco vai para uma transição, ou seja, o arco sai de um lugar e vai para uma transição, o arco aponta para uma transição, são chamados de lugares de entrada da transição, os lugares para os quais um arco sai de uma transição, ou seja, o arco sai de uma transição e vai para um lugar, o arco aponta para um lugar, são chamados de lugares de saída da transição. No exemplo acima, o lugar E_1 é uma entrada da transição T_1 e os lugares E_1 e E_2 são saídas da transição T_0 .

Graficamente, os lugares (*Places*) em uma rede de Petri podem conter um número discreto de fichas ou tokens, essas fichas indicam em que estado ou em que conjunto de estados o sistema modelado se encontra atualmente. Elas são representadas por um ponto dentro do lugar (*Place*) correspondente. Qualquer distribuição de tokens sobre os lugares (*Places*) representará uma configuração da rede chamada marcação.

Em sentido abstrato, uma transição de uma Rede de Petri pode disparar se ela for habilitada, ou seja, existem tokens suficientes em todas as entradas dos lugares (*Places*), quando a transição dispara, ela consome os tokens necessários da entrada e cria os tokens nos lugares (*Places*) de saída da transição. Um disparo é atômico, ou seja, ocorre em uma única etapa não interrompível.

Como exemplo, pode-se representar um semáforo utilizando as redes de Petri. Para isso, será criado três Lugares (*Places*), (representando cada um as luzes do semáforo, L_0 -verde, L_1 -amarelo e L_2 -vermelhos) e três Transições entre esses Lugares (*Places*) (representando as transições entre as luzes do semáforo, T_0 -verde para amarelo, T_1 -amarelo para vermelho, T_2 -vermelho para verde), conforme representado na Figura 16. Uma marcação (distribuição de um

token no Lugar (*Place*), *L0-verde*) será utilizada para indicar que o semáforo está atualmente no estado verde, amarelo ou vermelho.

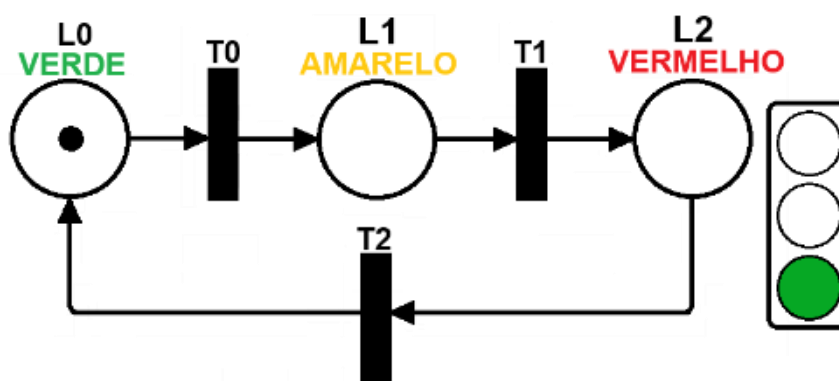


Figura 16 - Exemplo de modelagem de um semáforo com um token no estado verde.

É importante ressaltar que o estado amarelo não é indicado por uma transição, mas sim por um estado, a característica efêmera do estado amarelo pode dar a impressão de que ele é uma transição. No entanto, no modelo de redes de Petri, a transição é indivisível, ou seja, sem gradação visível.

Ao se observar o sistema em um determinado instante de tempo, ele estará em um estado ou em outro. O semáforo pode estar verde, vermelho ou amarelo; não existe estado perceptível entre dois desses estados. Ao realizar a afirmação que o semáforo amarelo é uma transição, o modelo se torna incapaz de representar uma dessas possibilidades. Pois as fichas só podem estar contidas dentro de lugares, não dentro de transições.

O estado do processo é expresso por uma distribuição do token entre os Lugares (*Places*) da Rede de Petri. O estado do processo pode ser descrito por vetores, e no exemplo acima o token distribuído no Lugar (*Place*) *L0*, pode ser descrito pelo vetor $(1, 0, 0)$.

Uma rede de Petri muda de um estado para o próximo quando uma transição é acionada/disparada (*Fired*). Um disparo (*Fire*) acontece quando uma transição ou um conjunto de transições são acionados de forma a alterar a marcação de uma Rede de Petri. O disparo (*Fire*) de uma transição envolve os lugares (*Places*) de entrada e saída da transição.

Os lugares (*Places*) de entrada para uma transição são todos aqueles lugares que têm uma aresta direcionada do lugar (*Place*) para a transição. Os lugares (*Places*) de saída de uma transição são todos aqueles lugares que têm uma aresta direcionada da transição para o lugar (*Place*). No exemplo acima, o lugar (*Place*) de entrada para a transição *T1* é o lugar *L1*. O lugar (*Place*) de saída para a transição *T0* também é o lugar *L1*, enquanto o lugar (*Place*) de saída

para a transição $T1$ é o lugar $L2$. O disparo (*Fire*) da transição $T0$ altera o estado do semáforo de verde para amarelo.

Uma transição somente pode ser disparada (*Fired*) se existirem marcações suficientes nos seus lugares (*Places*) de entrada, no exemplo acima, a transição $T1$ só é disparada se houver pelo menos uma ficha/token no lugar (*Place*) amarelo $L1$. Caso essa regra não seja satisfeita, será dito que o disparo resultou em uma exceção.

Em algumas redes de Petri os arcos entre transições e lugares possuem pesos. Se o peso estiver associado a uma entrada, ele indica a quantidade de fichas/tokens consumidos no disparo da transição. Se ele estiver em uma saída, ele indicará a quantidade de fichas/tokens criados após o disparo. Este trabalho, será restringido às redes de Petri com pesos unitários.

Em redes de Petri não-autônomas, o disparo pode ser habilitado por eventos externos (DAVID, ALLA, 2010). Dessa forma, pode-se associar cada transição a uma condição específica do sistema. No caso do semáforo, por exemplo, pode-se estabelecer uma condição de tempo mínimo para disparo de cada transição. Dessa forma, mesmo que o semáforo se encontre no verde, ele passará para o estado amarelo só após decorrer um tempo mínimo de espera.

A Rede de Petri possui uma grande capacidade e abrangência para representar sistemas complexos. Como exemplo a Figura 17 abaixo, ilustra essa capacidade ao se ampliar o exemplo anterior para dois semáforos que controlam a passagem de carros em um cruzamento em um ambiente de concorrência de recursos e de paralelismo.

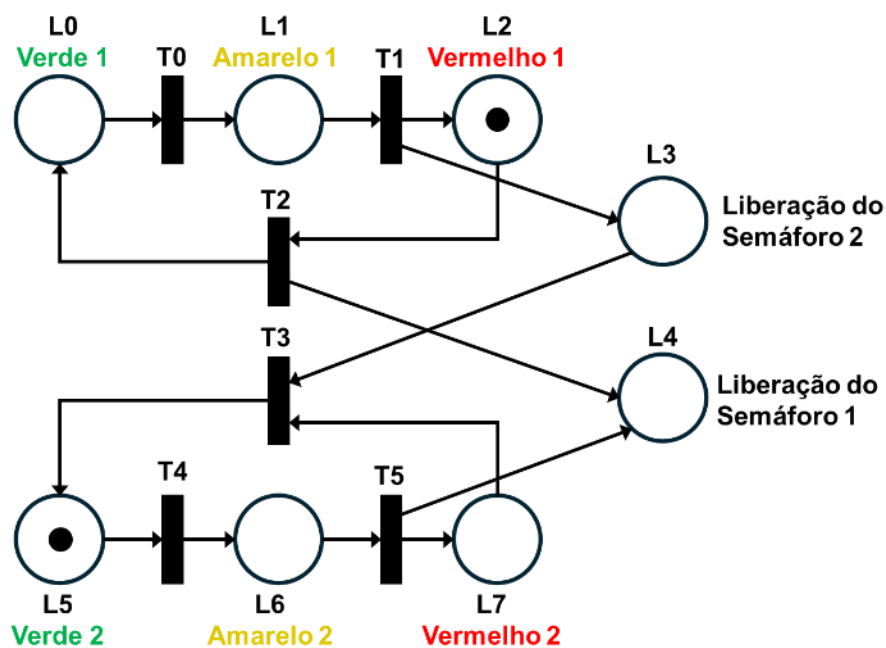


Figura 17 - Exemplo de tratamento de concorrência com Redes de Petri.

Nessa solução, cada semáforo é representado por uma marcação que percorre os estados verde, amarelo e vermelho. No entanto, para passar do estado vermelho para o verde, cada semáforo necessita de uma marcação de liberação, criada somente quando o outro semáforo passa da cor amarela para a vermelha.

2.8.1 Representação formal da Rede de Petri

Os lugares, transições e arcos de uma Rede de Petri são formalmente descritos pelas terminologia abaixo:

Uma Rede de Petri (PN) é definida por uma tupla de quatro elementos:

$$PN = (P, T, A, M_0) \quad (2.8.1.1)$$

Onde,

- P e T são conjuntos finitos de lugares e transições respectivamente, $P = \{p_1, p_2, \dots, p_n\}$, $T = \{t_1, t_2, \dots, t_m\}$, $(P \cap T = \emptyset, P \cup T = \emptyset)$
- A é o conjunto de arcos ou relações de fluxo, $A \subseteq (P \times T) \cup (T \times P)$
- $W: A \rightarrow \{1, 2, 3, \dots\}$, são os pesos dos arcos, que em regra é 1 para a_{ij} , ($a_{ij} \in A$, representa um arco vindo de p_i para t_j ou de p_j para t_i)
- M é o vetor de marcações (fichas/tokens) no conjunto de lugares (*Places*), $M = [M(p_1), M(p_2), \dots, M(p_n)] \in N^n$, M_0 é o vetor de marcação inicial.

Devido as marcações, a $PN = (P, T, A, M_0)$ é também chamada de Rede de Petri de P/T marcada.

Definição 1: disparando uma transição “ t ” em uma marcação “ M ” serão consumidos $W(p, t)$ tokens de cada um dos seus lugares de entrada “ p ”, e produz $W(t, p)$ tokens em cada um dos seus locais de saída “ p ”.

Definição 2: uma transição t_i é habilitada (pode disparar) no vetor de marcação M se houver tokens suficientes em seus locais de entrada para que os consumos sejam possíveis, ou seja, se e somente se,

$$\forall p \in P : W(p, t_i) \leq M(p) \quad (2.8.1.2)$$

ou seja, a transição t_i , só será habilitada apenas se todos os locais de entrada dela (lugares), possuírem uma quantidade de tokens que seja pelo menos igual ao respectivo peso do arco.

Definição 3: Os elementos p de entrada de uma transição t , denotados por “ $\bullet t$ ”, são o conjunto de todos os lugares que estão diretamente na entrada da transição t :

$$\forall p \in \bullet t = W(p, t) \neq 0 \quad (2.8.1.3)$$

Da mesma forma os elementos p de saída de uma transição t , denotados por “ $t \bullet$ ” são o conjunto de todos os lugares que estão diretamente ligados a saída de t :

$$\forall p \in t \bullet = W(p, t) \neq 0 \quad (2.8.1.4)$$

De forma similar os conjuntos de transições de entrada dos lugares e os conjuntos de transições de saída dos lugares, podem ser denotados por “ $\bullet p$ ” e “ $p \bullet$ ” respectivamente.

Definição 4: Quando uma transição habilitada t_i dispara, o novo vetor de marcação M_{j+1} é obtido do vetor de marcação anterior M_j da seguinte forma:

$$\forall p_i \in P, M_{j+1}(p_i) = M_j(p_i) - W(p_i, t_i) + W(t_i, p_i) \quad (2.8.1.5)$$

O disparo de uma transição t_i é denominada por $\vec{t}_i$.

Definição 5: A matriz de incidência e a equação de estado possibilitam a análise algébrica do comportamento dinâmico das redes de Petri. Para uma Rede de Petri $PN = (P, T, A, M_0)$ composta por “ m ” transições e “ n ” lugares, a matriz de incidência (A) é dada por:

$$A_{(n \times m)} = [a_{ij}^+ - a_{ij}^-] \quad (2.8.1.6)$$

onde, $a_{ij}^+ = W(t_j, p_i)$, representa o peso do arco indo da transição t_j para o lugar p_i , e $a_{ij}^- = W(p_i, t_j)$, representa o peso do arco indo do lugar p_i para a transição t_j , ($1 \leq i \leq n, 1 \leq j \leq m$), sendo que a_{ij}^- ou a_{ij}^+ é considerado zero se não existir arco do lugar p_i para a transição t_j e vice versa, e o elemento da matriz de incidência A tem um valor -1 quando o lugar p_i for uma entrada da transição t_j , e tem um valor 1 quando o lugar p_i for uma saída da transição da transição t_j .

Definição 6: A equação (2.8.1.7) é a equação de estado de uma Rede de Petri P/T Marcada $PN = (P, T, A, M_0)$. Ela apresenta um novo vetor de marcação devido ao disparo do vetor $f(t)$ (vetor de disparo).

$$M' = M_0 + A^T \times f(t) \quad (2.8.1.7)$$

Onde A^T é a matriz de incidência transposta. O vetor coluna de estado M' gerado pela equação de estado devido a uma série de disparos de $f(t)$ no estado inicial M_0 é denotado pelo operador $M_0[f(t)]M'$.

Seja $\sigma = t_{i1} t_{i2} \dots t_{ik}$ (onde $t_{ij} \in T, j \in \{1, 2, \dots, k\}$) uma sequência de disparo de transição, tem-se que σ está habilitado em relação a M se $M[t_{i1}]M_1[t_{i2}]M_2 \dots M_{k-1}[t_{ik}]$, onde $M_1, M_2, \dots, M_{k-1}$, são marcações intermediárias com entradas inteiras não negativas; isso é indicado por $M[\sigma]$.

Sendo $M[\sigma]M_k$ a indicação de que o disparo σ de M produz M_k e $\bar{\sigma}(t)$ o número total de ocorrências da transição “ t ” em σ . Mais especificamente, $\bar{\sigma} = [\bar{\sigma}(t_1) \bar{\sigma}(t_2) \dots \bar{\sigma}(t_m)]^T$ é o vetor de disparo que corresponde a σ . O vetor de disparo de uma única transição $t_i, 1 \leq i \leq m$ é indicado por $\bar{t}_i$, ou seja, por um vetor coluna m -dimensional com uma única entrada diferente de zero com valor “1” em seu i -ésimo lugar. A sequência de disparo de transição vazia é indicada por “ ε ” e seu vetor de disparo correspondente é $\bar{0}_m$, um vetor coluna m -dimensional com todas as entradas definidas como zero.

Note que após disparar uma sequência de transições σ da marcação M , a marcação final M_k pode também ser computada como $M_k = M + A\bar{\sigma}$; nesse caso, tem-se que a marcação M_k é alcançável pela marcação M via uma sequência de disparo σ . Dois vetores de disparo $\bar{\sigma}'$ e $\bar{\sigma}$ de mesma dimensão podem ser comparados elemento a elemento. Mais especificamente, usa-se $\bar{\sigma}' \preceq \bar{\sigma}$ para indicar que $\bar{\sigma}' \leq \bar{\sigma}$ (elemento a elemento) e $\bar{\sigma}' \neq \bar{\sigma}$. Note que $\leq$ é apenas uma relação de ordem parcial neste caso e existem $\bar{\sigma}'$ e $\bar{\sigma}$ que não são comparáveis. Por exemplo, os vetores de disparo $\bar{\sigma}' = [1 \ 1 \ 2]^T$ e $\bar{\sigma} = [2 \ 2 \ 0]^T$ não são comparáveis porque os dois primeiros elementos de $\bar{\sigma}'$ são menores do que aqueles de $\bar{\sigma}$ enquanto o terceiro elemento é maior do que o terceiro elemento de $\bar{\sigma}$. Em outras palavras, eles não satisfazem a condição $\bar{\sigma}' \leq \bar{\sigma}$ ou $\bar{\sigma}' \neq \bar{\sigma}$ (elemento a elemento).

2.8.2 Representação gráfica de um exemplo de Rede de Petri

A Rede de Petri tem uma grande capacidade para representar diversos sistemas caracterizados como sendo concorrentes, assíncronos, distribuídos, paralelos, não determinísticos e/ou estocásticos (SILVA, 2017). Para ilustrar essa capacidade será feito a representação de um processo de teste de usinagem de uma peça fabricada, em Redes de Petri, conforme Figura 18.

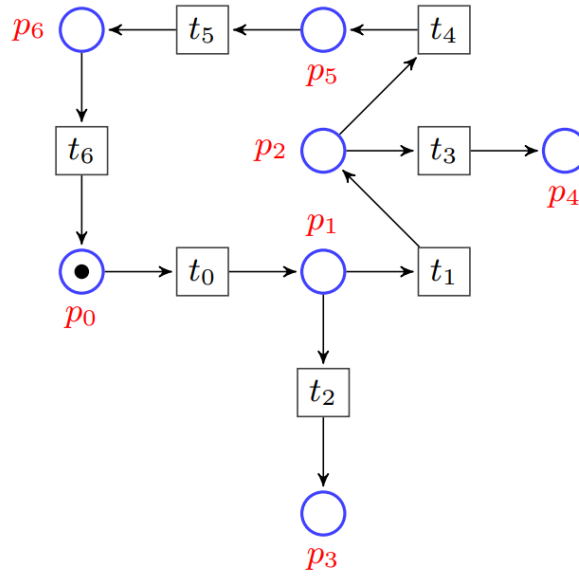


Figura 18 - Esquema de teste de peça usinada. Fonte: *Improving Exception Analysis in PNRD systems using Labelled Free Choice Nets*

Na representação acima, existem 7 transições $T = \{t_0, t_1, t_2, t_3, t_4, t_5, t_6\}$, 7 lugares, $P = \{p_0, p_1, p_2, p_3, p_4, p_5, p_6\}$ e um token em p_0 .

Conforme descrito tem-se que: $M_0 = [1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]^T$

A matriz de incidência transposta pode ser representada por:

$$A^T = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

Para encontrar o próximo vetor de marcação M_k é necessário efetuar o disparo em t_0 . Conforme definido na equação (2.8.1.2), a transição t_0 está habilitada no vetor de marcação M_0 , pois $\forall p \in P : W(p, t_0) \leq M(p)$.

Seguindo a equação 2.8.1.7, $M' = M_0 + A^T \times f(t)$, e sendo $f(t_0) = [1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]^T$, tem-se:

$$M_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (2.8.2.1)$$

2.8.3 PNML

PNML - *Petri Net Markup Language* (Linguagem de marcação de rede de Petri), é um formato de intercâmbio de mensagens baseado em XML para Rede de Petri, que visa permitir que ferramentas de Redes de Petri troquem modelos de Rede de Petri. PNML é uma proposta para traduzir as representações matemáticas e gráficas da Rede de Petri em forma de um arquivo baseado em XML.

Tecnicamente, o modelo básico de uma PNML é um pacote de linguagem de modelagem unificada (UML-*Unified Modeling Language*), havendo pacotes UML adicionais para os diferentes tipos de Rede de Petri. O Framework PNML é uma implementação livre e de código aberto da ISO/IEC-15909, *International Standard on Petri Nets*. Ele foi projetado para dar suporte ao padrão ISO/IEC 15909-2, e permite que desenvolvedores de ferramentas de redes de Petri integrem em suas ferramentas o padrão PNML, bem como fornece uma API Java abrangente e compreensível para criar e salvar modelos de Rede de Petri em arquivos PNML e carregar e buscar modelos de rede de Petri em arquivos PNML.

Há muitas ferramentas para modelagem de Rede de Petri, cada ferramenta possui uma forma própria de gerar os arquivos PNML. O Framework PNML, veio para ajudar os desenvolvedores de ferramentas a implementarem uma interface para PNML, fornecendo uma estrutura e uma API para carregar, buscar e salvar documentos de rede de Petri em arquivo PNML.

A PNML foi projetada para ser um formato de intercâmbio de Rede de Petri que é independente das ferramentas e plataformas utilizadas. O formato de intercâmbio precisa suportar diferentes dialetos de Redes de Petri e deve ser extensível a várias plataformas.

A PNML deve ser capaz de representar qualquer tipo de Rede de Petri com suas extensões e recursos específicos, considerando-a como um gráfico rotulado, onde todas as informações adicionais podem ser armazenadas em rótulos que podem ser anexados à própria rede, aos nós da rede ou aos seus arcos, o tipo específico da PNML deve ser determinado exclusivamente a partir de sua representação e a descrição deve ser inequívoca.

A Figura 19 mostra as diferentes partes e relacionamentos da PNML. O módulo de Metamodelo define a estrutura básica de um arquivo PNML, a Interface de Definição de Tipo permite a definição de novos tipos de Rede de Petri que restringem os arquivos legais do Metamodelo, o módulo de Interface de Definição de Funcionalidades permite a definição de

novos recursos para Redes de Petri, o módulo de Convenções de Documentos contém a definição de um conjunto de recursos padrões de Redes de Petri, que são definidos de acordo com a interface de definição de recurso.

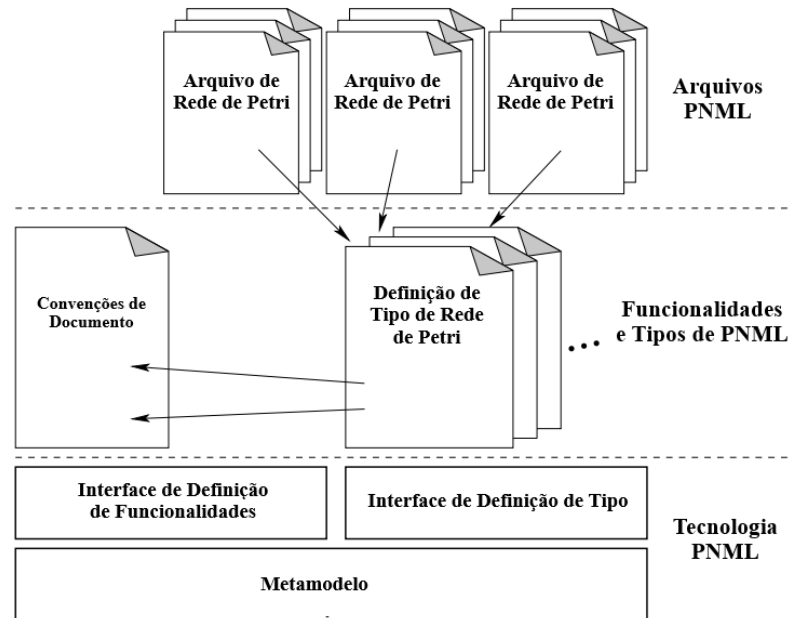


Figura 19 - Visão geral da estrutura PNML. Fonte: https://www.researchgate.net/figure/Overview-of-the-parts-of-PNML_fig1_220783478, acessado em 20/09/2024

A Figura 20 mostra a estrutura do módulo Metamodelo da PNML em linguagem de modelagem unificada (UML-*Unified Modeling Language*).

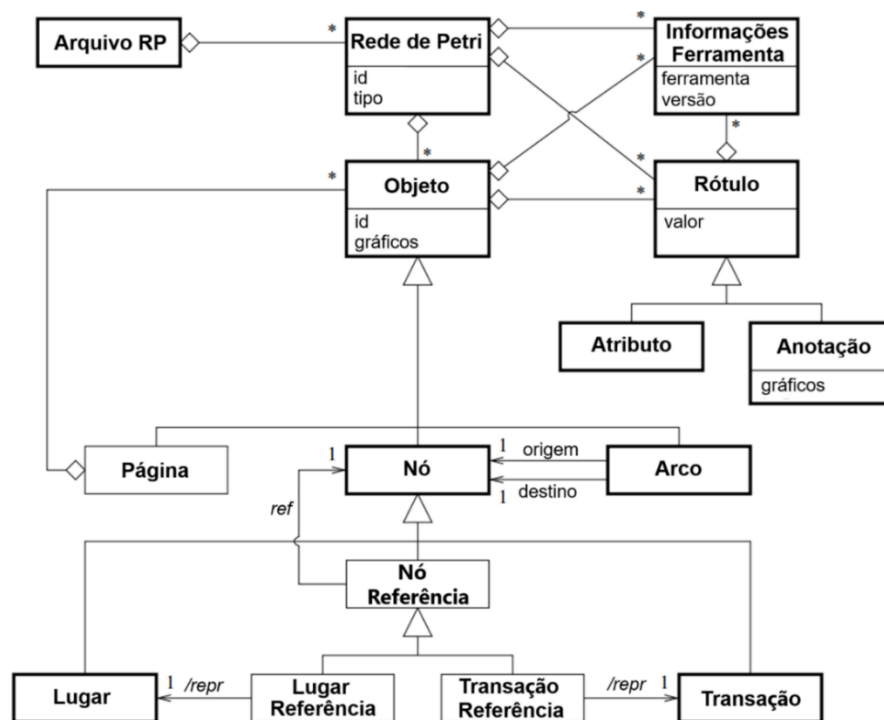


Figura 20 - Estrutura do módulo Metamodelo da PNML. Fonte: https://www.researchgate.net/figure/The-PNML-meta-model_fig2_220783478, acessado em 20/09/2024

Um arquivo de Rede de Petri (RP) é aquele que atende aos requisitos da PNML e pode conter várias redes de Petri, cada Rede de Petri consiste em objetos, que, basicamente, representam a estrutura do gráfico da Rede de Petri, cada objeto (lugar, transição ou arco) dentro de um arquivo de rede de Petri tem um identificador exclusivo, e um nó pode ser um lugar ou uma transição.

2.8.4 PNRD (Rede de Petri Inserida em Base de Dados RFID) e iPNRD (PNRD invertida)

PNRD (*Petri Net Inside RFID Database* - Rede de Petri Inserida em Base de Dados RFID) é uma estrutura formal de dados baseada em Redes de Petri Elementares, que armazena equações de estado de Redes de Petri em Etiquetas (Tags) RFID (marcação M_k e matriz adjacente A^T) e em Leitores (*Readers*) RFID (vetor de disparo u_k). (TAVARES; SARAIVA, 2010)

Dessa forma, a PNRD é uma estrutura de dados formal cujas informações da Rede de Petri estão separadas e armazenadas nos componentes do sistema RFID, Tags e Leitores/Antenas, conforme ilustra a Figura 21.



Figura 21 - PNRD integrando Redes de Petri e RFID

Na PNRD ocorre a união do mapeamento de processos (conforme a modelagem em Redes de Petri), e a tecnologia RFID, para que seja possível a aplicação do modelo matemático de uma Rede de Petri em uma estrutura física composta por Tag, Leitor e Antena. A matriz de incidência transposta A^T e o vetor de marcação inicial M_k , juntamente com indicação do tempo são enviados para a Tag e o vetor de disparo u_k é enviado para o Leitor/Antena de forma a poder calcular o valor do próximo estado durante a captura dos dados da Tag. O Leitor/Antena, além de armazenar o vetor de disparos, se comunica com a aplicação e outros dispositivos e realiza os disparos. A PNRD invertida (iPNRD) surge como uma variação da PNDR comum, onde a lógica de armazenamento das informações da equação do vetor de marcação é invertida, o vetor de disparos u_k são gravados nas etiquetas e a matriz de incidência transposta A^T , o vetor de marcação M_i ficam no Leitor/Antena, sistema e controladores. Dessa forma as etiquetas podem ser adicionadas a agentes ativos do meio, enquanto o leitor pode estar presente em agentes

passivos. Uma aplicação para a iPNRD foi proposta e desenvolvida por Fonseca e Tavares (2018), onde a iPNRD seria utilizada em busca e salvamento de pessoas em trilhas de caminhada, buscando-se assim viabilizar a utilização do conceito da PNRD em aplicações em que a instalação de antenas RFID nos pontos de transição seja demasiadamente difícil.

A Figura 22 apresenta como a PNRD distribui os fatores da equação de estados nos componentes RFID. No caso da PNRD invertida ou iPNRD, a tag armazena o vetor de disparo u_k enquanto o leitor retém a matriz de incidência transposta A^T e o vetor de marcação inicial M_k .

O processo funciona por meio do protocolo NDEF - *NFC Data Exchange Format*, a Tag ao entrar na área de um Leitor/Antena, recebe um comando de requisição de informações, envia a informação para o Leitor/Antena, que recebe os dados, decodifica e realiza os cálculos da Rede de Petri, atualizando o vetor de marcação M_k em função do cálculo da equação de estado. Caso o resultado seja adequado o status da operação será NO_ERROR, e é feita a gravação do vetor de marcação M_k na Tag, caso não ocorra nenhum problema no processo de gravação o Leitor/Antena envia o status NO_ERROR para a aplicação conectada ao Leitor/Antena. Esse processo se repete até o fim de todas as etapas mapeadas.

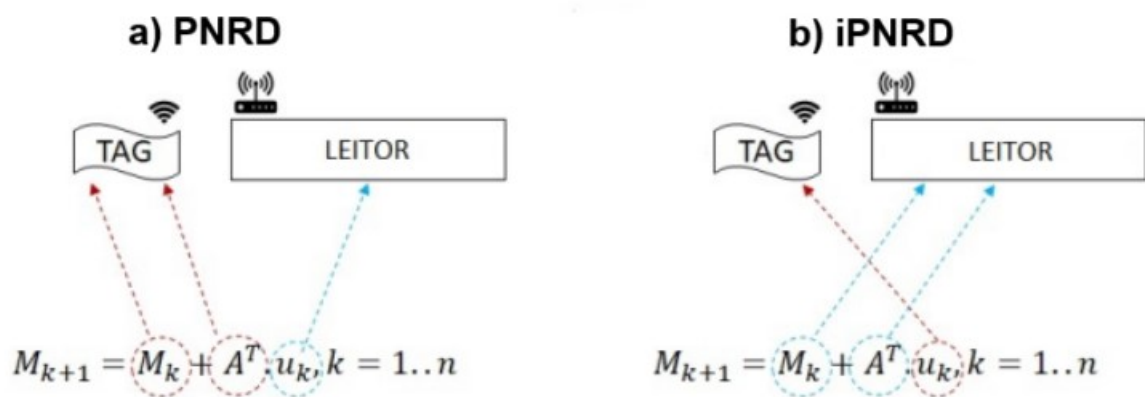


Figura 22 - Associação dos termos da equação do vetor de marcação com os elementos armazenados na PNRD (a) e iPNRD (b). Fonte: FONSECA et al., 2018

Uma característica importante da PNRD são as exceções, a ocorrência de um erro implicará em um valor negativo no vetor de marcações, e será imediatamente identificada pelo Leitor/Antena RFID. Abaixo são listadas as exceções para os disparos da rede PNRD, conforme padronização adotada pela biblioteca PNRD (SILVA, 2017).

Módulo PetriNet:

- NO_ERROR – o disparo foi realizado e o vetor de marcação atualizado sem erros.

- `CONDITIONS_ARE_NOT_APPLIED` – as condições associadas às transições do disparo não estão satisfeitas;
- `PRODUCE_EXCEPTION` – o disparo resulta em exceção, ou seja, não existem marcações suficientes nos lugares de entrada para efetuar as transições
- `NOT_KNOWN` – o programa segue por um fluxo inesperado.

Módulo PNRD:

- `NO_ERROR` – a operação de escrita ou leitura foi efetuada com sucesso;
- `TAG_NOT_PRESENT` – não existe Tag RFID compatível com o leitor na sua proximidade;
- `INFORMATION_NOT_SAVED` – exclusivo para o caso de escrita. Indica que ocorreu um erro no ato de gravação dos dados na etiqueta;
- `INFORMATION_NOT_PRESENT` – exclusivo para o caso de configuração inicial, indica que nem todas as informações configuradas estão presentes na etiqueta lida.
- `DATA_SIZE_NOT_COMPATIBLE` – exclusivo para o caso de leitura, significa que o objeto PNRD criado não alocou uma quantidade de memória adequada para representar as informações contidas na etiqueta;
- `NOT_ENOUGH_SPACE` – exclusivo no caso de gravação, indica que a etiqueta não tem memória interna suficiente para gravar os dados;
- `NOT_AUTHORIZED` – significa que o ato de escrita ou leitura não foi autorizado pela etiqueta;
- `VERSION_NOT_SUPPORTED` – refere-se ao fato de que o leitor não suporta a versão de dados presentes na etiqueta;
- `ERROR_UNKNOWN` – significa que houve a ocorrência de um erro genérico não especificado.

A ideia principal do PELMS é intermediar o modelo de Rede de Petri com um sistema automatizado de logística e manufatura. O processo alvo é associado a um modelo comportamental pretendido, representado formalmente por Redes de Petri.

Um exemplo de aplicação do modelo é a produção de mancais por uma fábrica, conforme a Rede de Petri da Figura 23 abaixo, onde após uma peça ser usinada, é necessário a inspeção da peça para verificar se ela atende os padrões de cilindricidade e rugosidade do seu furo. Caso o teste de cilindricidade falhe, a peça é enviada para a reciclagem, se o teste de

rugosidade falhar ela é colocada em uma fila para ser ajustada e se os dois testes tiverem sucesso, a peça é armazenada em estoque.

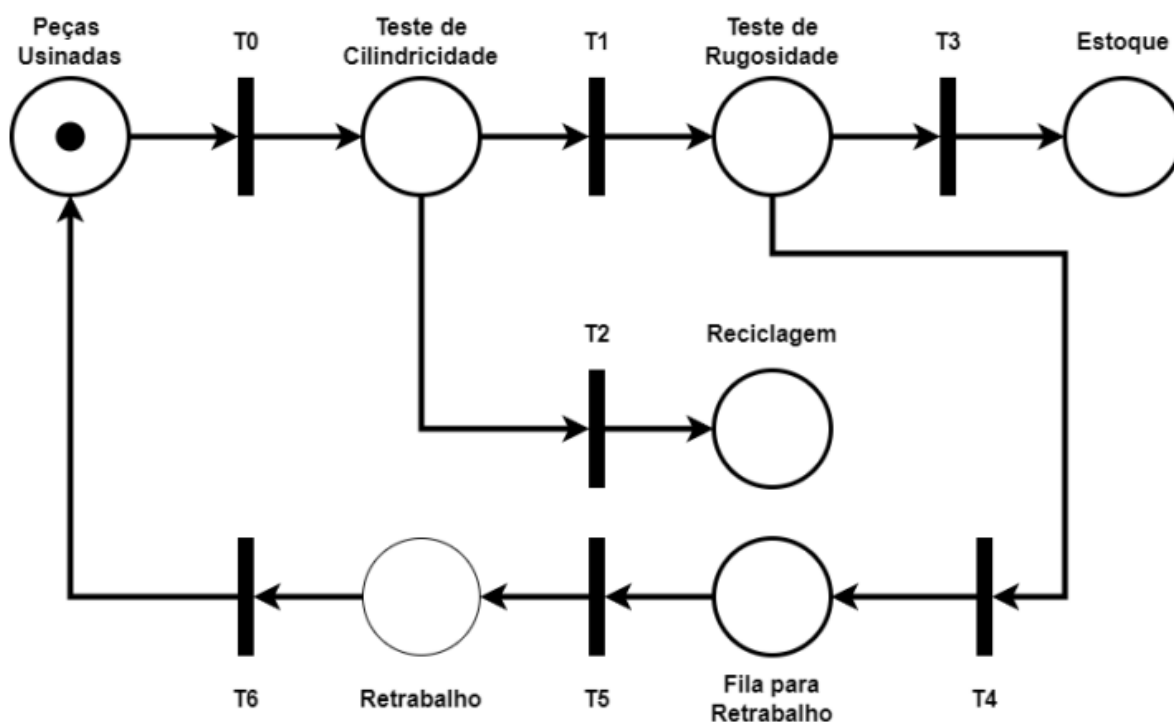


Figura 23 - Exemplo de uma PNRD para verificação de mancais. Fonte: SILVA (2017)

Cada peça a ser usinada contém uma Tag RFID com um id de identificação, contendo a matriz de incidência da Rede de Petri, o vetor de marcações inicial e a indicação de tempo. A indicação de tempo é incluída de forma a possibilitar o sistema a realizar análises quantitativas de performance. Em cada etapa de produção as máquinas de verificação devem conter leitores RFID para verificação/leitura das etiquetas.

Assim quando uma nova peça se aproximar, o sistema irá coletar as informações contidas na etiqueta da peça (vetor marcação e matriz de incidência), realizar o disparo da transição T0, calcular o próximo estado do sistema e gravá-lo na etiqueta. Em seguida, de acordo com o novo estado da etiqueta (teste de cilindricidade), o sistema envia um comando para começar o teste de cilindricidade, e dependendo do resultado do teste, o sistema envia uma mensagem ao Sistema de Controle conectado aos Leitores/Antena RFID posicionados nos Lugares da RP, que por sua vez dispara a transição T1 ou T2 e atualiza o novo estado da etiqueta. Em seguida a peça é enviada para a reciclagem ou para o teste de rugosidade. Se o resultado desse disparo for uma exceção, percebe-se que a peça não foi enviada para o lugar correto, e da mesma forma, pode-se detectar peças não-conformes ou não testadas enviadas para o estoque.

2.9 ESP32

Um dos principais microcontroladores utilizados para desenvolvimento de soluções IOT são os ESP, desenvolvido pela empresa ESPressif System, uma empresa Chinesa, que produz microcontroladores e sistemas em chips, eles possuem uma grande vantagem em relação a uma gama de outros microcontroladores, por possuir módulos para comunicação wireless embutidos no próprio chip, e outras vantagens que os destacam entre os disponíveis no mercado, neste meio a família ESP32 apresenta-se, entre as linhas de produto da Expressif®, como uma das mais completas do mercado.

O ESP32 é uma placa microcontroladora do ecossistema Arduino, famoso pela sua eficiência de processamento, baixo custo e baixo consumo de energia. Oferece suporte a diversas tecnologias de conexão, incluindo Wi-Fi, Bluetooth, ZigBee, tornando-o altamente versátil. Equipado com um processador dual-core 32bits de até 240 MHz, 520 KB de RAM e 4/8/16 MB de memória flash interna, Antena PCB na placa ou conector para antena externa, além de uma variedade de periféricos como UART, SPI, I2C e câmera, foi totalmente desenvolvido para projetos de IoT (ESP32).

Apesar dos ESPs serem entendidos como uma placa de desenvolvimento, às vezes até como uma alternativa semelhante ao Arduino, é importante fazer a distinção clara entre os dois componentes. O Arduino é uma plataforma de prototipagem eletrônica de código aberto, onde tem-se vários Circuito Integrado (CI) com diferentes funções, e no caso do ESP32 que é um SoC - System on Chip (Sistema em Chip), ele é responsável por todas estas capacidades acima. Por outro lado, o fabricante disponibiliza em seu catálogo diversos modelos (Figura 24), sendo eles:

- ESP32 SoCs DOWN/SOWD/U4WD e PICO: Chip base utilizado no ESP32, com as nomenclaturas D/S O WD R2 H Q6 V3, (D/U = dual core, S = single core; O/2/4 = 0, 2 ou 4MB de flash memory; WD = Wi-Fi b/g/n + Bluetooth/Bluetooth LE dual mode; R2 = PSRAM 2MB; H = high temperature; Q6 = encapsulamento QFN 6x6 o padrão é o QFN 5x5; V3 = Revisão do chip), é o próprio SoC, sendo este o modelo mais simples e econômico para inclusão em produtos já de longo ciclo no mercado, porém utilizar este componente diretamente no circuito exige maior habilidade e um maior tempo dedicado ao design da interface, uma vez que ele traz consigo uma série de *guidelines* a serem seguidos para sua instalação.
- ESP32-WROOM: Esses são módulos baseados em ESP32-DOWD com flash integrado. Esses módulos são bem adequados para aplicativos de conectividade baseados em

Wi-Fi e Bluetooth/Bluetooth LE e fornecem um desempenho dual-core sólido. Este módulo é de fácil inclusão no design de outras PCBs, já possui proteção térmica e de ruído, circuito para filtro da fonte de tensão, gerador de clock e antena PCB inclusos.

- Série ESP32-WROVER: uma série baseada no ESP32-D0WD SoC, tendo também memória flash e SPIRAM integrados, ideal para trabalhos que precisam atingir um ótimo desempenho, é implementada com um chip dual-core e é bem adequada para aplicações que exigem mais memória, como AIoT e aplicações de gateway.
- Série ESP32-MINI, baseada no ESP32-U4WDH, possui memória flash integrada, fornecendo assim uma solução econômica para aplicações de conectividade simples baseadas em Wi-Fi e Bluetooth/Bluetooth LE.
- ESP32-DevKitC: é uma placa de prototipagem com uma pinagem de simples acesso para as saídas e entradas no módulo, de forma que o uso de jumpers facilita a utilização, bem como possui botões para mudar o chip para o modo de gravação, porta USB, e Leds indicadores de funcionamento.

São disponibilizados pelo fabricante outras opções de soluções resultante da combinação dos modelos acima, possibilitando o desenvolvimento de componentes que atenda as demandas do mercado.



Figura 24 - Modelos de Microcontroladores ESP. Fonte: <https://www.vidadesilicio.com.br>, acessado em 22/09/2024

Embora também seja aplicável em áreas como robótica, entretenimento, automação residencial, automotiva e agora industrial, sua flexibilidade de linguagens é um enorme diferencial na comunidade, permitindo aplicação em uma ampla gama de projetos além dos voltados para IoT. Graças à sua capacidade de programação em linguagens como C/C++ e sua compatibilidade com a Arduino IDE, o ESP32 tornou-se uma escolha popular entre os desenvolvedores.

Na Figura 25 pode-se observar o mapa de pinos presentes no ESP-WROOM-32 DEVKIT V1, juntamente com as especificações das entradas e saídas.

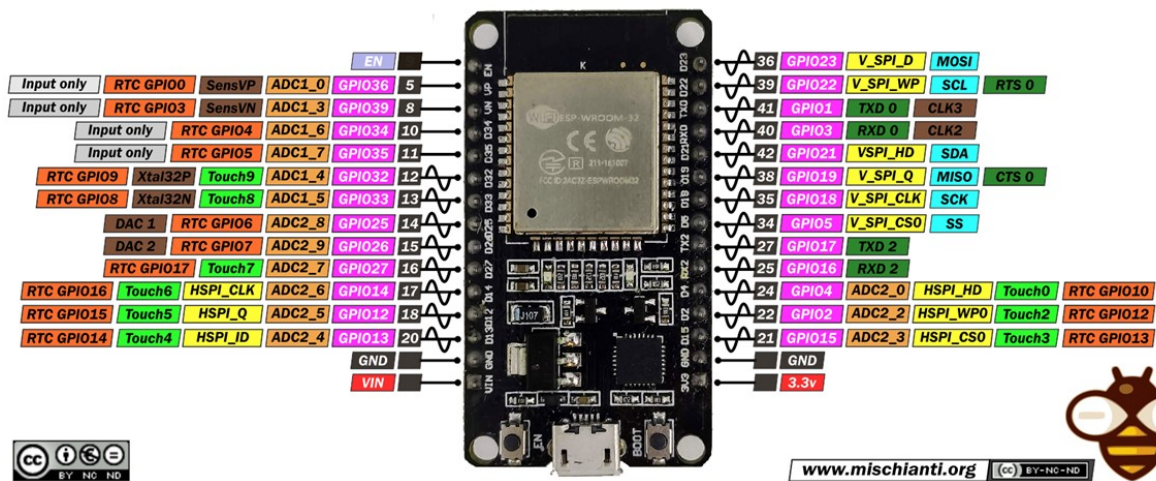


Figura 25 - Mapa de Pinos ESP-WROOM-32 DEVKIT V1. Fonte: <https://mischiati.org>, acessado em 21/09/2024

Outra característica importante dos ESP32 é que eles utilizam o ESP-IDF o ambiente de desenvolvimento oficial disponibilizado pelo fabricante baseado FREERTOS®. Um ambiente amplamente documentado com utilização do Cmake para organização do processo compilação e inclusão das bibliotecas e, em nível mais alto, de scripts Python para automatização de processos como reconhecimento da placa, link e upload, assim como demonstrado na Figura 26. Além disso, do ponto de vista da compilação o ESP-IDF utiliza versões oficiais do GNU *Compiler Collection* (GCC) com *backend* específico para a família de processadores Xtensa® LX63; e são, portanto, compatíveis com todas as linguagens suportadas no *frontend* do GCC utilizado (C, C++, *Objective-C*, FORTRAN, Go etc.).

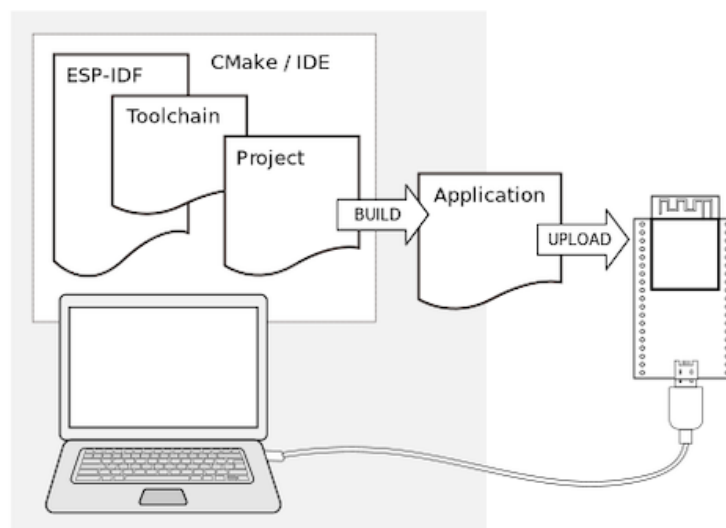


Figura 26 - Modelo ESP-IDF. Fonte: <https://docs.espressif.com>, acessado em 22/09/2024

Ao utilizar o ESP32 pela primeira vez, deve-se instalar o drive do ESP32, conectando a placa ESP no computador via cabo micro USB, o computador não irá reconhecer corretamente o dispositivo da porta do ESP, e não disponibilizará a interface para uso. A placa ESP32 DEVKIT V1 DOIT e muitos outros módulos ESP32 usam os chips CP2102 ou CP2104, e para instalar os drivers CP210x corretos, é necessário obter o driver CP210x USB para UART Bridge Virtual COM Port na página da Silicon Labs (<https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers?tab=downloads>), para então instalar o driver correto.

O microcontrolador ESP32 pode ser usado com diversos IDEs-*Integrated Development Environment* (Ambiente Integrado de Desenvolvimento), tais como ESP-IDF, Arduino IDE, etc. Os IDE é um ambiente integrado de desenvolvimento, escrito em funções de linguagem de programação, que oferece uma série de recursos para programar as placas de várias categorias, facilitando a gravação de códigos no microcontrolador da placa, ele é usado para interagir com placas de microcontroladores, permitindo escrever e fazer upload de programas nas placas compatíveis.

Neste trabalho foi adotado o IDE Arduino 2.0, por ser a melhor escolha e possuir flexibilidade para trabalhar com diversos modelos de microcontroladores, desempenho aprimorado, interface de usuário aprimorada e muitos outros recursos, como preenchimento automático, depurador integrado e sincronização de esboços com o Arduino Cloud.

2.10 Raspberry PI

A fundação Raspberry Pi é uma instituição de caridade educacional com sede no Reino Unido (Silva, 2019), onde desenvolveram dispositivos voltados para o aprendizado de crianças em programação em Python. Esse dispositivo foi denominado como Raspberry Pi. Do tamanho de um cartão de crédito ele é um computador capaz de inúmeras atividades, como navegar na internet, reproduzir vídeo, manipular planilhas, processar textos, compilar e executar programas, e jogar jogos (Silva, Dias et Escudero, 2022). Além dessas atividades, ele pode interagir com dispositivos externos como sensores, atuadores e de comunicação por meio da conexão GPIO (Entrada e Saída de uso geral).

De acordo com a organização Raspberry Pi, não há nada que o computador faça que o Raspberry Pi não consiga realizar (Silva, 2019). Tornando o Raspberry Pi útil tanto para projetos pequenos e simples quanto para projetos grandes e complexos, como nas indústrias, casas inteligentes e Internet das Coisas (IoT). A fundação Raspberry Pi fábrica diferentes series

de computadores, sendo elas *Flagship* (Principal), *Zero*, *Compute Module* (Módulo de Computação) e *Pico* (Raspberry Pi, 2024).

- A série *Flagship* ou Principal oferece hardware de alto desempenho e um sistema operacional Linux completo. Ele é dividido em dois modelos A e B, o modelo A é mais barato e vem com algumas alteração de hardware como a remoção da porta ethernet, e diminuição do tamanho da memória RAM e da quantidade de porta USB, já o modelo B ele é completo. Os modelos de Raspberry Pi que se encontram na série *Flagship* são: Raspberry Pi Model B, Raspberry Pi Model A, Raspberry Pi Model B+, Raspberry Pi Model A+, Raspberry Pi 2 Model B, Raspberry Pi 3 Model B, Raspberry Pi 3 Model B+, Raspberry Pi 3 Model A+, Raspberry Pi 4 Model B, Raspberry Pi 400 e Raspberry Pi 5; (Informações sobre a série *flagship* encontra-se no link <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#schematics-and-mechanical-drawings>)

- A série *Zero* oferece um sistema operacional Linux completo e portas essenciais a um preço acessível. Eles são divididos em dois modelos, os que já vem com o pino soldado (sufixo H) e os pinos que precisam ser soldados na placa (sem sufixo H). Os modelos de Raspberry Pi que se encontram na série *Zero* são: Raspberry Pi Zero, Raspberry Pi Zero W, Raspberry Pi Zero WH e Raspberry Pi Zero 2 W; (Informações sobre a série *Zero* encontra-se no link <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#schematics-and-mechanical-drawings>)

- A série *Compute Module* oferece hardware de alto desempenho e um sistema operacional Linux completo, possui um formato ideal para aplicações industriais. O que o difere dos anteriores é a falta de pinos GPIO e menor número de portas. Os modelos de Raspberry Pi que se encontram na série *Compute Module* são: Raspberry Pi *Compute Module* 1, Raspberry Pi *Compute Module* 3, Raspberry Pi *Compute Module* 3+, Raspberry Pi *Compute Module* 4S e Raspberry Pi *Compute Module* 4; (Informações sobre a série *Compute Module* encontra-se nos links <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#schematics-and-mechanical-drawings> e <https://www.raspberrypi.com/documentation/computers/compute-module.html>)

- A série *Pico* não roda o sistema operacional Linux e não permite armazenamento removível, mas permite a programação e carregamento para a placa. Eles são divididos em dois modelos, os que já vem com o pino soldado (sufixo H) e os pinos que precisam ser soldados na placa (sem sufixo H). Os modelos de Raspberry Pi que se encontram na série *Pico* são: Raspberry Pi Pico, Raspberry Pi Pico H, Raspberry Pi Pico W, Raspberry Pi Pico WH e

Raspberry Pi Pico 2. (Informações sobre a série Pico encontra-se nos links <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#schematics-and-mechanical-drawings> e <https://www.raspberrypi.com/documentation/microcontrollers/pico-series.html>)

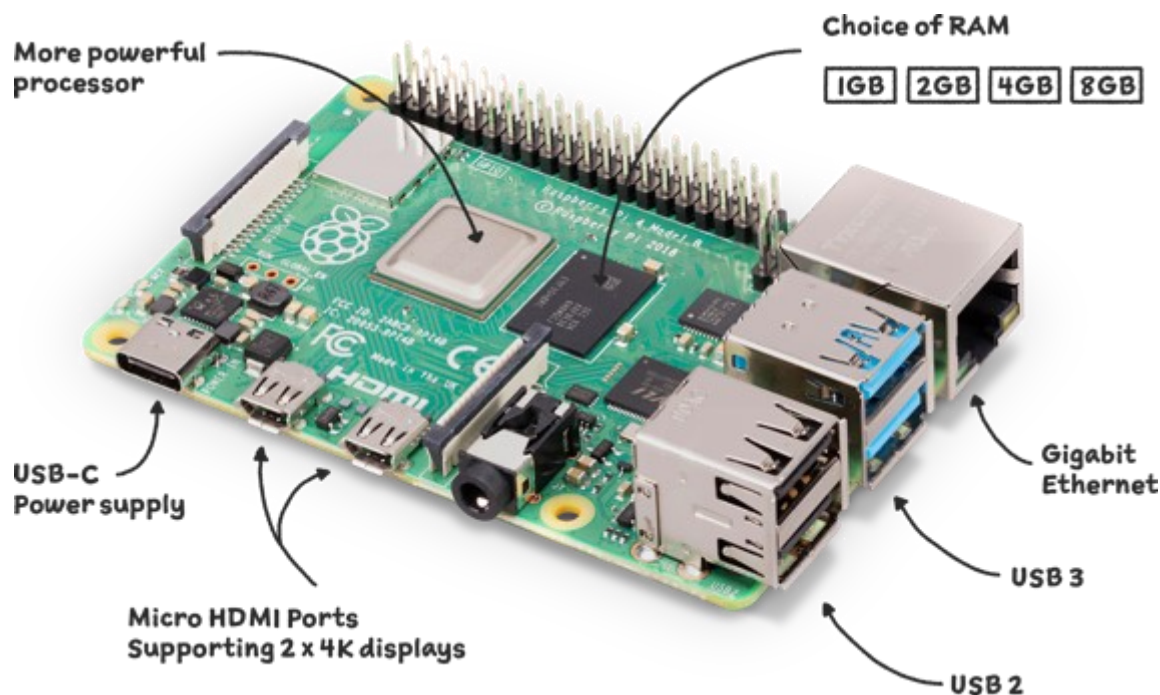


Figura 27 - Raspberry Pi 4 Model B. Fonte: <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/>, acessado em 22/09/2024

A Figura 27 ilustra o Raspberry Pi 4 Model B, que será utilizado neste trabalho, sendo suas características:

- Dimensões totais: 85 mm x 56 mm;
- SoC (Sistema em um Chip – System-on-a-Chip) BCM2711 (Raspberry Pi, 2024):
 - Processador: Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5 GHz;
 - Memória Cache: 32 kB de dados + 48 kB de instruções de Cache L1 por core, e 1 MB de Cache L2;
 - Multimídia: H.265 (4Kp60 decodificador), H.264 (1080p60 decodificador, 1080p30 encode) e OpenGL ES com gráficos de até 3.0;
- Memória RAM: SDRAM LPDDR4 8GB 2.4 GHz;
- GPIO: 40 pinos GPIO header
- Entradas/Saídas (I/O):
 - 2 micros HDMI;
 - 2 USB 2.0 e 2 USB 3.0;
 - Conector AV de 3.5 mm;

- Conector Camera Pi;
- Conector Display;
- Gigabit (1Gb/s) Ethernet RJ45;
- Slot de cartão Micro SD;
- *Wireless Driver:*
 - WiFi 2.4/5 GHz dual-band 802.11 ac (120 Mb/s);
 - Bluetooth 5.0 com BLE (Bluetooth Low Energy)
- Alimentação: entrada USB tipo C com carga de entrada 5 V e 3 A (15 W).

Inicialmente o Raspberry Pi desenvolveu um sistema operacional baseado na Kernel do Linux, denominado Raspbian. No entanto, devido ao crescente uso e popularidade dessa plataforma, surgiram diversas outras distribuições de Sistemas Operacionais Linux específicos para o gerenciamento da RPi (Silva, Dias et Escudero, 2022). Já para os modelos a partir da RPi 3 é possível executar o sistema operacional Windows 10 IoT Core Insider Preview, lançado em 2015 pela Microsoft para ser usado em dispositivos como Raspberry Pi, Arduino e o Minnow Board (Microsoft, 2016).

O sistema operacional Raspbian é uma derivação da distribuição Debian projetado e compilado para a plataforma Raspberry Pi. Ele oferece mais de 35.000 pacotes “.deb” de software, que estão pré-compilados para serem facilmente instalados nesta distribuição de sistema operacional. O Raspbian embarca o ambiente de desktop LXDE (*Lightweight X11 Desktop Environment*), o gerenciador de janelas *OpenBox*, o navegador Midori, as ferramentas para desenvolvimento de software e o código fonte de exemplo de funções multimídia.

Dentre as plataformas de prototipagem, tais como, Raspberry Pi®, Arduino® e ESP32®, foi escolhido o Raspberry Pi 4 Model B para o projeto, por ser uma plataforma amplamente divulgada e com grande volume de material de estudo disponível, além de dispor de integração e comunicação entre seus hardwares e periféricos e possuir múltiplos sistemas operacionais, GUI (Interface Gráfica de Usuário -*Graphical User Interface*), mouse, teclado, Wi-Fi etc., desta forma, focou-se no desenvolvimento de programas e softwares, ao invés de implementar soluções para que o hardware funcionasse adequadamente.

2.11 MQTT

MQTT (Transporte de Telemetria de Enfileiramento de Mensagens – *Message Queuing Telemetry Transport*) é um protocolo de comunicação para a Internet das Coisas, criado pela

IBM na década de 90 com foco em atender à necessidade de um protocolo simples e leve, facilitando a comunicação de forma segura dos microcontroladores para a obtenção de dados com uma taxa de transmissão leve e com qualidade de serviço. Segundo a OASIS® (Organização para o Avanço de Padrões em Informação Estruturada – *Organization for the Advancement of Structured Information Standards*) o protocolo MQTT é ideal para ambientes restritos, como no contexto de comunicação Máquina para Máquina (M2M – *Machine to Machine*) e Internet das Coisas (IoT), onde poucas linhas de código e pouca largura de banda é necessária.

O protocolo MQTT tem ganhado espaço em diversas aplicações por se mostrar um modelo em que cada componente é capaz de controlar o tipo de informação que será transmitida e/ou recebida, o que é visto como um grande ganho do ponto de vista de IoT, além disso, a entrada e saída de novos componentes em rede acontece de forma bastante simples, sem necessitar que o responsável pela rede refaça toda sua configuração. O protocolo MQTT é baseado no protocolo TCP/IP (protocolos de comunicação entre computadores, TCP - *Transmission Control Protocol* – Protocolo de Controle de Transmissão e IP - Internet Protocol – Protocolo de Internet) e ambos, cliente e broker, necessitam da pilha TCP/IP para o seu funcionamento.

O MQTT está na mesma camada OSI que o HTTP, porém a maior diferença entre os dois protocolos é o tamanho do payload. No HTTP - *Hypertext Transfer Protocol* - Protocolo de Transferência de Hipertexto, o payload é maior, o que inviabiliza o seu uso em conexões de baixa qualidade, além disso, o MQTT possui maior segurança (mais níveis de serviço), menor complexidade e permite comunicação de 1 para N dispositivos.

A Figura 28 representa o modelo OSI (Sistema Aberto de Interconexão - *Open System Interconnection*) para alguns tipos de protocolos de comunicação IoT. O protocolo MQTT opera na camada de aplicação do modelo OSI e pode ser integrado com protocolos de transmissão que forneçam dados ordenados, sem perdas, e conexões bidirecionais, como o TCP/IP. Este protocolo utiliza o modelo de mensagem publicador/subscritor (*publisher/subscriber*), que utiliza o princípio de publicar mensagens (*Publisher*) e subscrever tópicos (*Subscriber*) para receber mensagens, isto implica que o publicador e o subscritor não trocam dados diretamente, mas realizam a interação de mensagens de forma indireta, por meio do intermediário chamado Broker, ele é responsável por intermediar as mensagens entre as máquinas (ENGPROCESS).

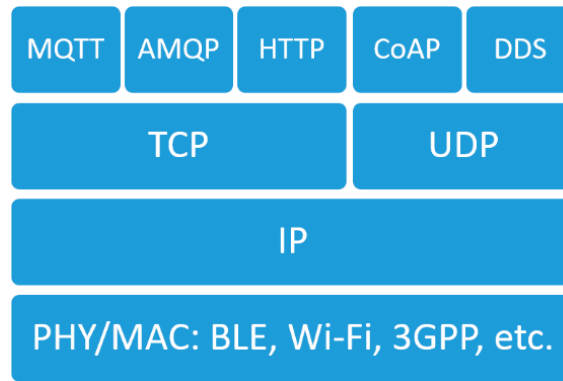


Figura 28 - Modelo OSI de alguns protocolos de comunicação IoT. Fonte: <https://www.researchgate.net>, acessado em 27/09/2024

O protocolo MQTT é um protocolo de transferência de publicação/assinatura de camada de aplicação, e para funcionar utiliza a arquitetura *publisher/subscriber* (envio/recebimento de mensagens) em que são definidos uma série de tópicos e cada componente pode se cadastrar como publicador de um grupo de tópicos e subscritor de outro grupo de tópicos (grupos não exclusivos), na Figura 29 é possível ver a representação de um sistema onde o celular e servidor secundário para *Backend* consultam dados publicados pelo sensor de temperatura. Assim, os servidores desta rede (ou Brokers) podem deixar a função de orquestradores da rede e se tornarem responsáveis por rotear as mensagens transmitidas por cada tópico. Com o uso do MQTT, há um grande avanço no sentido de objetos em rede autogerenciáveis, mas informações sobre o uso desta ferramenta em específico neste trabalho serão apresentadas mais à frente.

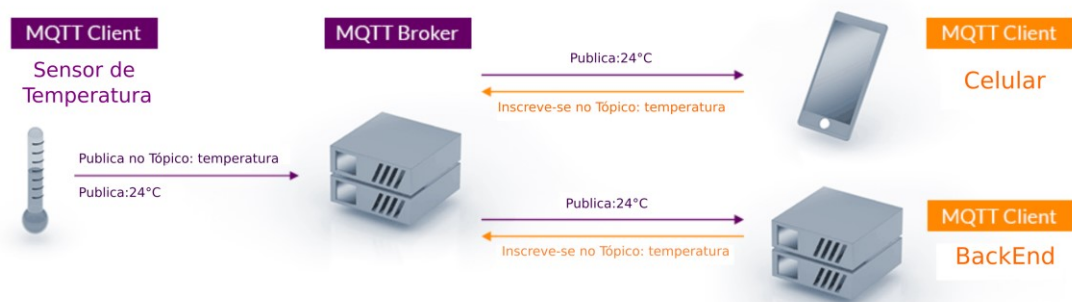


Figura 29 - Diagrama representativo da arquitetura MQTT. Fonte: <https://alfacomp.net/>, acessado em 20/10/2024

O Broker, que é responsável por filtrar as mensagens e saber exatamente para quem enviar. Dessa forma, o *publisher* e *subscriber* não precisam se conhecer diretamente e apenas precisam conhecer o Broker, que é quem fará a notificação da mudança de estados e enviará essa informação para aqueles que tiverem inscritos no tópico referenciado. Por fim, o Publisher precisa se preocupar apenas com enviar as informações e estabelecer a conexão exclusivamente com o Broker.

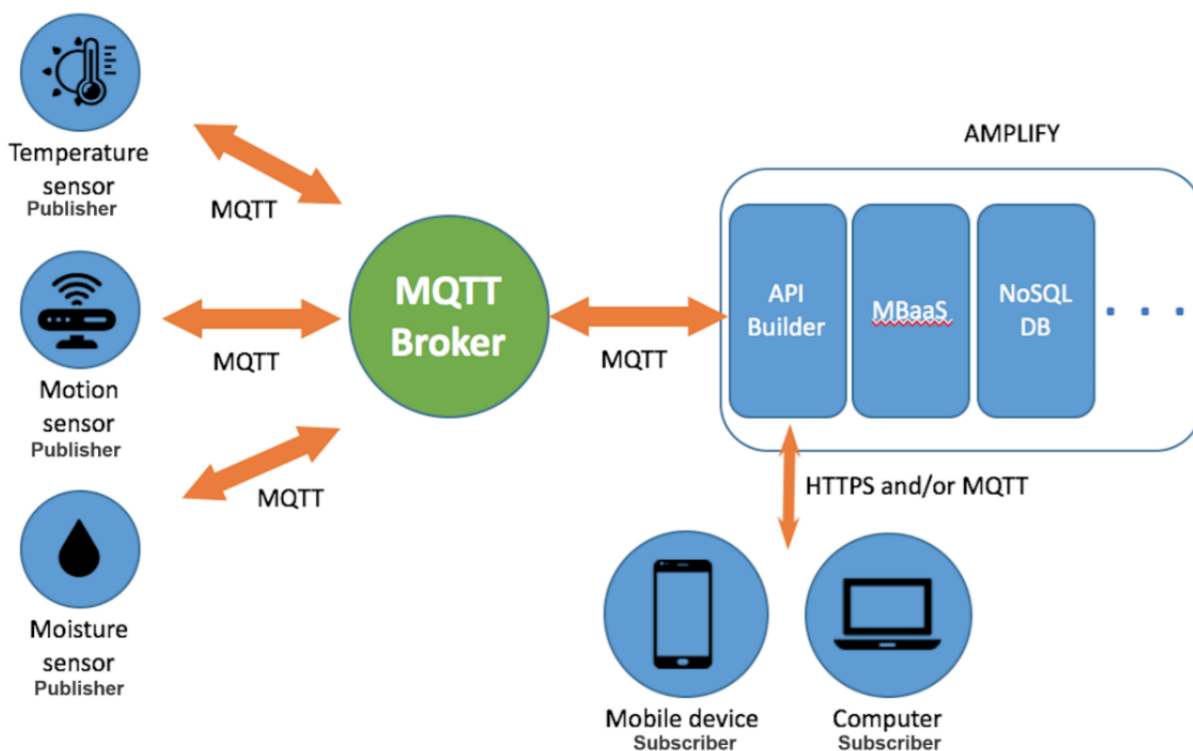


Figura 30 - Infraestrutura de Comunicação do MQTT com o Broker, Publisher e Subscriber. Fonte: <https://alfacomp.net/>, acessado em 20/10/2024

No MQTT, os tópicos são configurados entre o cliente e o servidor, que são então assinados e publicados. Para relatórios de sensor, o cliente do sensor se registra em um servidor para publicar dados do sensor e vice-versa se registra para tópicos para receber atualizações quando necessário. Os cabeçalhos MQTT são de um Byte no mínimo. Campos opcionais podem aumentar o tamanho do cabeçalho quando necessário.

O protocolo oferece suporte à qualidade de serviço de publicação com base na identificação do QoS (Qualidade de Serviço – *Quality of Service*). Durante o processo de publicação, o cliente armazena a mensagem no *payload* (pacote de mensagem) do publicador e envia uma mensagem ao Broker de acordo com o QoS definido pelo usuário. Existem três tipos de qualidade de serviço para entrega de mensagens.

A primeira é o QoS=0, que significa entrega no máximo uma vez. Quando o Broker recebe uma mensagem *PUBLISH* do usuário, ele a encaminha para todos os assinantes inscritos neste tópico. Nesses dois processos, o *payload* é enviado apenas uma vez e não tem a necessidade de recebimento de um ACK de confirmação.

Quando QoS=1, significa pelo menos uma vez. Após receber a mensagem *PUBLISH* do usuário, o Broker retorna uma mensagem de confirmação *PUBACK*. Em seguida, a mensagem *PUBLISH* pode ser enviada a todos os assinantes inscritos no tópico.

Quando QoS=2, significa exatamente uma vez, o Broker registra a mensagem *PUBLISH* recebida do usuário, mas só a envia a todos os assinantes após receber a mensagem *PUBREL*, o Broker envia a mensagem *PUBLISH* para todos os assinantes inscritos no tópico e envia uma mensagem *PUBCOMP* ao usuário.

Pelo lado do subscritor/assinante, inicialmente, o cliente estabelece uma conexão TCP com o servidor e envia uma mensagem *CONNECT*. Após receber a mensagem de confirmação *CONNACK* autorizada pelo Broker, o cliente envia uma mensagem *SUBSCRIBE*, especificando a lista de tópicos de interesse. Por fim, o servidor retorna uma mensagem de confirmação *SUBACK*, indicando que a assinatura foi bem-sucedida (Souza, R. et al.2022).

O MQTT usa o TLS como comunicação segura, para um Broker local, quando um usuário tenta conectar ele necessita de um usuário e senha pré-cadastrados no Broker quando iniciado, mas mesmo sendo uma comunicação segura o Broker ainda será capaz de saber a mensagem enviada (Santos, P. et al. 2020). Por conta de todas essas qualidades este protocolo é perfeito para uma comunicação M2M, com dados leves, robusto, e em tempo real é perfeito para a comunicação entre máquinas no setor industrial. Por exemplo, um sensor e um atuador, ligados fisicamente na alimentação, conectados ao Broker via Wi-fi, o sensor publicando as informações coletadas, logo a frente no processo, um atuador subscrevendo a informação do sensor no tópico atua de acordo com o desejado.

A Figura 30 acima, mostra esquematicamente uma comunicação entre um sensor via MQTT que pode ser acessado via mobile ou computador. O Broker do protocolo serve como um servidor armazenando tópicos, podendo ser exemplificado como sendo blocos de memória, cada tópico criado é um bloco de memória e nele é possível armazenar uma informação (*payload*), um usuário ou uma máquina pública uma mensagem em um tópico, outro usuário ou máquina acessa esse tópico fazendo um *Subscribe*, de acordo com a mensagem buscada a máquina ou o usuário fara a devida tratativa e ação.

Desenvolvido pela Eclipse Foudation® o Broker Eclipse Mosquitto, ou somente Mosquitto, é uma implementação de um Broker MQTT com interface de simples aprendizagem, tanto para instalações locais, quanto para aquelas feitas de forma remota. Além disso, o Mosquitto é conhecido por ser um programa que requer para seu funcionamento poucos recursos seja no âmbito do processamento ou no âmbito do espaço em memória.

Embora muito simples, o Broker Mosquitto possui uma série de funções importantes que vão desde segurança via certificados x509 até a implementação do modo *Bridge*: Um modo

de funcionamento onde as mensagens publicadas em um Broker são repetidas em um ou mais espelhos, sendo o contrário também possível, todas estas possíveis configurações podem ser consultadas na documentação dos arquivos ".

Quando se fala em comunicação via Internet uma das primeiras preocupações que surgem em qualquer projeto é quanto à segurança e integridade dos dados trafegados pela rede, e quando se fala de utilização de servidores em nuvem, esta preocupação fica ainda maior, visto que os dados serão tratados e armazenados seguindo as políticas de terceiros que muitas das vezes podem não ser compatíveis com os requisitos desejados. Atualmente, técnica mais utilizada para garantir a segurança na transmissão de dados é a utilização do Transfer Layer Security (TLS), uma série de protocolos definidos no RFC 5246 para o processo de encriptação utilizando, em partes criptografia de chave simétrica e em outros momentos criptografia de chaves assimétricas, podendo ser dividido em dois grandes protocolos:

- TLS *Record Protocol* utilizado diretamente para a transferência de dados, que serão criptografados geralmente seguindo o modelo definido na *Advanced Encryption Standard* (AES) tendo como chave simétrica o valor definido ao final do processo de negociação.
- TLS *Handshake Protocol* onde são definidas as regras para a negociação da forma como será feita a transferência dos dados (algoritmo de criptografia, algoritmo de *hash*, tamanho de bloco, *padding*, entre outros). Vale ressaltar que nesta etapa deve ser utilizado algum algoritmo de chaves assimétricas como *Rivest-Shamir-Adleman* (RSA) ou *Elliptic Curve Digital Signature Algorithm* (ECDSA) para garantir a identidade dos negociantes.

O *Socket Secure Layer* (SSL), por sua vez, possui grande semelhança com o protocolo TLS inclusive na maneira como é redigido o documento emitido pela Netscape® em 1996 revisado em 2011 "*The SSL Protocol - Version 3.0*", possuindo praticamente a mesma estrutura. Porém, o desenvolvimento dos métodos utilizados na Internet e principalmente da segurança da informação mostraram que algumas alterações eram necessárias para a manutenção da segurança nas transmissões, assim limitando a utilização de algumas funções de *Hash*, métodos de geração de números pseudoaleatórios, incluindo passos intermediários e entre outros.

3 METODOLOGIA E DESENVOLVIMENTO

3.1 Estudos Iniciais

Os estudos iniciais para a implementação do PELMS-MQTT se iniciaram durante o estágio obrigatório, onde foi feito a revisão de conceitos e aplicações de Redes de Petri inseridas em banco de dados RFID em Arduino® e ESP32®, e posteriormente teve continuidade nos períodos do PFC I e II. Inicialmente foi realizado a preparação de um material explicativo sobre a Biblioteca Arduino para PNRD (da Silva, 2017). Em seguida, foi feito os testes da biblioteca na plataforma Arduino®, onde, após compreender e confirmar o funcionamento da PNRD no Arduino® foi definido em qual software seria implementado/testado a biblioteca PNRD para plataforma ESP32®.

Confirmado o funcionamento da biblioteca PNRD no ESP32® e observando algumas falhas de funcionamento da mesma, foi necessário um estudo dos códigos contidos na biblioteca Petri Net, PNRD, Pn532NfcReader, PN532, PN532_HSU e NDEF para aprimoramento do funcionamento da PNRD. Posteriormente foi feito o estudo do funcionamento do PALMS, onde não foram encontradas informações relacionadas ao desenvolvimento do software, sendo necessário o estudo dos códigos contidos nos projetos PALMS-FTP (Carrijo, 2021) e PALMS-MQTT (Pereira, 2022).

Após os estudos foi identificado os arquivos principais para o funcionamento do middleware, “PalmsGui.py” e “MainWindow.py”, realizando algumas alterações e algumas complementações no *middleware* PALMS-MQTT foi possível gerar arquivos *template* para serem inseridos tanto no *reader* quanto no recorder, a atualização dos dados e os disparos que podem ser inseridos nas TAGs, e o retorno do resultado dos disparos via MQTT para o middleware PELMS-MQTT, permitindo o funcionamento do sistema em tempo real. Outra melhoria é a possibilidade do middleware funcionar com os sistemas operacionais Linux e Windows.

3.2 Instalação do Sistema Operacional no Raspberry PI 4

A instalação do sistema operacional no Raspberry Pi 4 Model B deve ser feita em um cartão SD armazenando dentro dele a imagem do sistema operacional Raspbian ou qualquer outro sistema operacional compatível com a plataforma. A ferramenta utilizada para o armazenamento da imagem do sistema operacional no cartão SD foi o Raspberry Pi Imager (Figura 31). Nessa ferramenta poderá ser escolhido o modelo de Raspberry Pi, o sistema

operacional e o dispositivo de armazenamento que será utilizado, em seguida, é possível realizar alguns setups, como o *hostname*, o *username* e o *password*, a conexão WLAN e o SSH.

Após o armazenamento da imagem do sistema operacional no cartão SD, deverá ser inserido no Raspberry Pi para que ele realize a instalação do sistema operacional. Instruções da instalação do sistema operacional podem ser encontradas no link <https://www.raspberrypi.com/documentation/computers/getting-started.html>. O sistema operacional escolhido para esse trabalho foi o Raspbian devido a compatibilidade dele com o sistema operacional Debian.

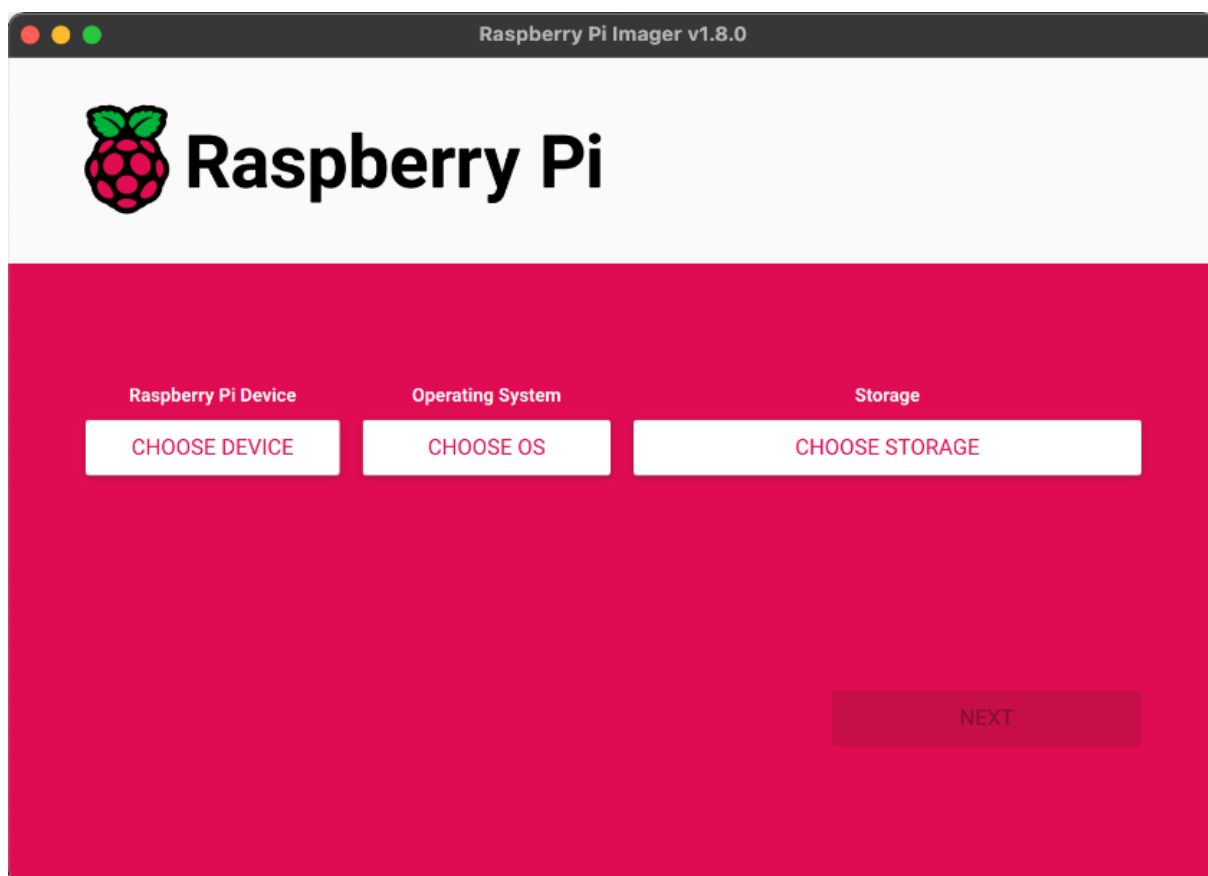


Figura 31 - Software de geração da imagem dos sistemas operacionais disponíveis para instalação no Raspberry Pi. (Acessado em 07/12/2024: <https://www.raspberrypi.com/documentation/computers/getting-started.html>)

3.3 Instalação e Configuração do Broker

A instalação e o setup do Broker foram realizados na plataforma Raspberry Pi 4 Model B, onde foi instalado o sistema operacional Raspbian, e no servidor Debian 12 disponível no Laboratório de Planejamento de Automação e Manufatura (MAPL – *Manufacturing Automated Planning Lab*). Como um dos objetivos do trabalho foi selecionado para instalação tanto no Raspberry Pi 4 Model B, quanto no servidor Debian 12 o Broker Mosquitto MQTT, onde ele será responsável por armazenar e gerenciar a comunicação MQTT utilizando o protocolo

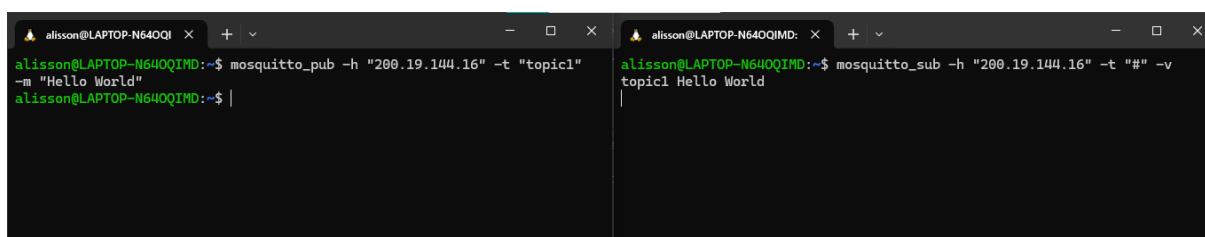
MQTT. Os passos de instalação do Broker em uma distribuição Linux se encontram no APÊNDICE A.

Após a instalação do Broker, deverá ser feito o setup (configuração) do Broker para que os dispositivos conectem a ele sem identificação utilizando a porta 1883. Como complemento foi feito o setup de um bridge entre o Raspberry Pi 4 Model B e o servidor Debian 12 no laboratório MAPL. O bridge serve para manter os dois servidores atualizados e com dados referentes aos processos. Feito a instalação e a configuração, o Broker pode ser inicializado. Os passos de setup e inicialização do Broker do servidor MAPL e do Raspberry Pi 4 Model B se encontram no APÊNDICE B.

3.4 Comunicação Publicador-Subscritor

Após a instalação do Raspberry Pi e o setup do Broker foram feitos alguns testes de comunicação utilizando o protocolo MQTT para publicação e subscrição no Broker. Em seguida, foram testados manualmente o funcionamento do middleware PELMS-MQTT, onde ele publicou/subscreveu corretamente as informações e carregou/registrou as informações em arquivos. Após o middleware atender aos requisitos do projeto, foram feitos os testes de publicação e subscrição das plataformas ESP32® *reader/recorder* para o armazenamento/disparo da TAG.

Os testes foram feitos no sistema operacional Ubuntu 20.04 (Linux) embarcado no Windows 10 (Utilizando a ferramenta WSL2) e a plataforma Arduino IDE. Para o primeiro teste foi aberto duas janelas do terminal Ubuntu (Figura 32), onde na primeira janela, foi configurada para receber mensagens de qualquer tópico, e na segunda janela, foi feito a publicação do tópico “*topic1*” com a mensagem “*Hello World*” e na primeira janela, foi recebido a mensagem do “*topic1*” “*Hello World*”, que havia sido armazenada temporariamente no Broker.



```
alisson@LAPTOP-N64OQI:~$ mosquitto_pub -h "200.19.144.16" -t "topic1" -m "Hello World"
alisson@LAPTOP-N64OQI:~$ |

alisson@LAPTOP-N64OQIMD:~$ mosquitto_sub -h "200.19.144.16" -t "#" -v
topic1 Hello World
```

Figura 32 - Teste de Subscrição e Publicação no Broker.

O segundo teste foi dividido em três etapas, a primeira etapa é a passagem dos dados referente ao disparo, a segunda etapa é a passagem dos dados referente ao recorder e a terceira etapa é a subscrição dos dados referente aos disparos ocorridos na rede de Petri.

Na primeira etapa foi feito a configuração do cliente MQTT para subscrição de todos os tópicos e apresentação das mensagens recebidas na janela de terminal do Ubuntu, em seguida foi aberto o arquivo “*setup.pelms*”, através da opção do menu do middleware, “*File>Open Setup File (*.pelms)*”, e pressionado o botão “*TRANSFER PNRD SETUP*” da aplicação PELMS-MQTT, para a leitura das informações da rede de Petri contidas no arquivo “*pnrpData.pnr*” ou “*ipnrpData.pnr*” e envia as informações utilizando os tópicos “*PELMS-MQTT/fire/len_place*”, “*PELMS-MQTT/fire/len_transition*”, “*PELMS-MQTT/fire/incidence_matrix*”, “*PELMS-MQTT/fire/markings_vector*” e “*PELMS-MQTT/fire/fire_vector*”, que são apresentados na janela do terminal Ubuntu (Figura 33).

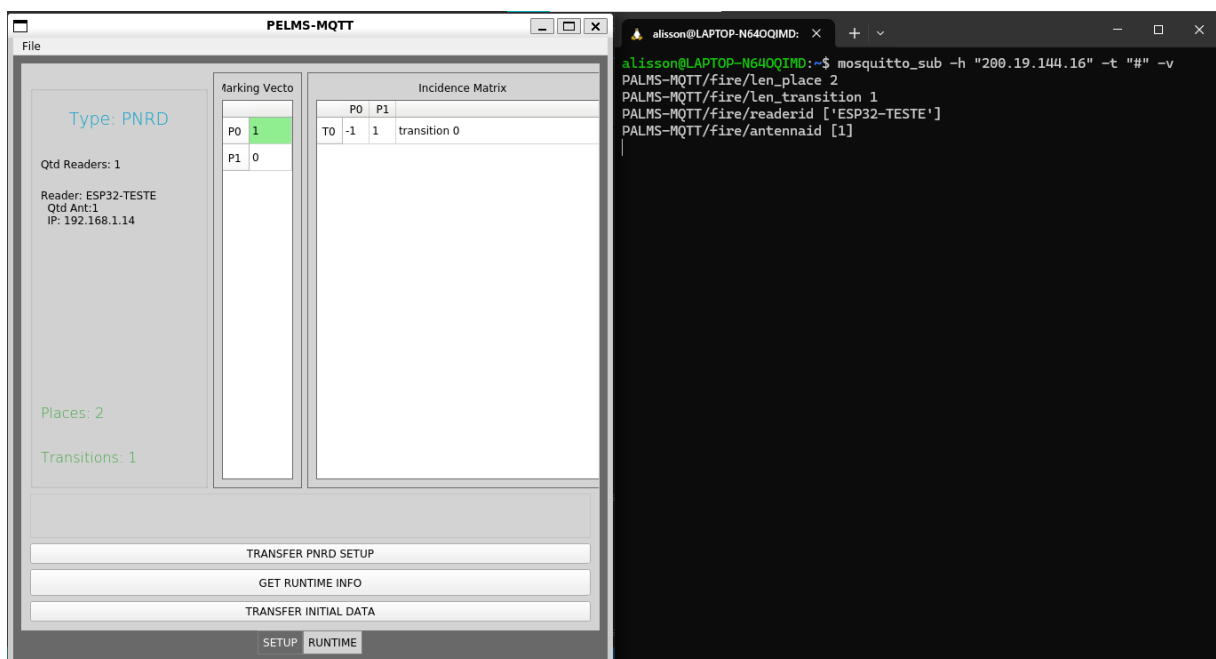


Figura 33 - Publicações dos Tópicos do ESP32® Disparador.

Na segunda etapa, foi pressionado o botão “*TRANSFER INITIAL DATA*” da aplicação PELMS-MQTT, onde o middleware faz a leitura das informações contidas no arquivo e as envia utilizando os tópicos “*PELMS-MQTT/len_place*”, “*PELMS-MQTT/len_transition*”, “*PELMS-MQTT/incidence_matrix*”, “*PELMS-MQTT/markings_vector*” e “*PELMS-MQTT/fire_vector*”, que são apresentados na janela do terminal Ubuntu (Figura 34).

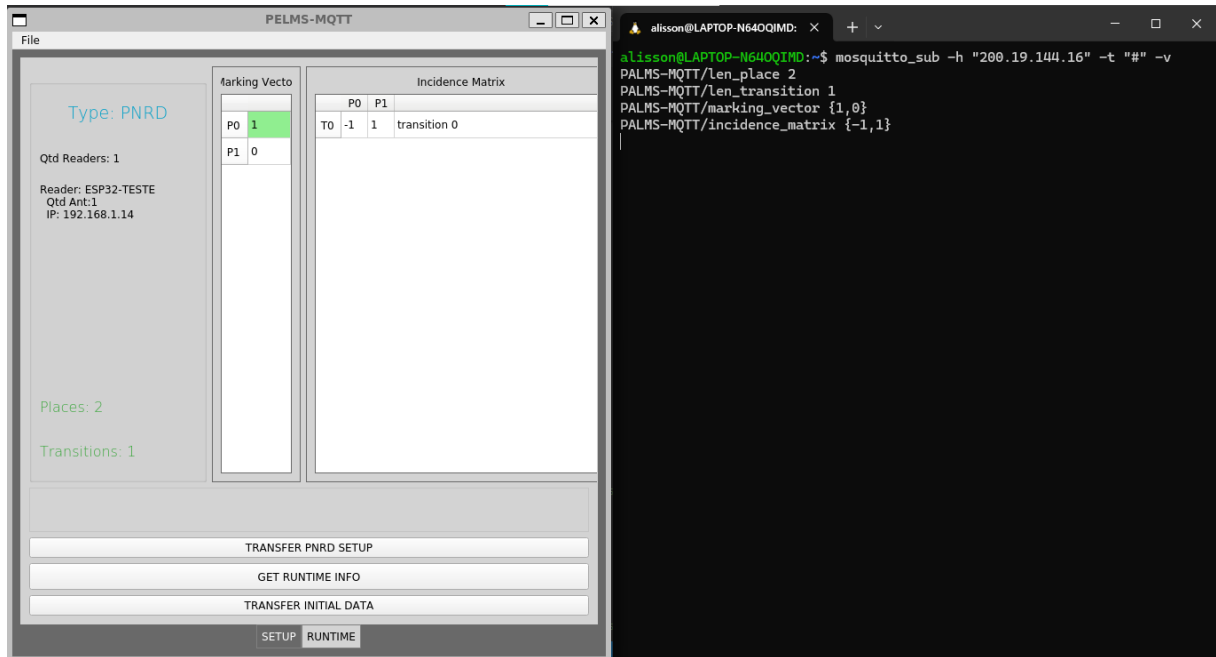


Figura 34 - Publicações dos Tópicos do ESP32 Recorder.

Na terceira etapa é feita a liberação do middleware para recebimento de mensagens via MQTT (pressionando o botão “*GET RUNTIME INFO*”), o que faz com que o sistema subscreva todos os tópicos com o rótulo iniciado por “*PELMS-MQTT/Runtime/*”, quando o middleware receber um evento de disparo, as informações enviadas nos tópicos serão coletadas e armazenadas em um arquivo “*runtime.pnrd*”, juntamente com a data e a hora do recebimento da mensagem (Figura 35).

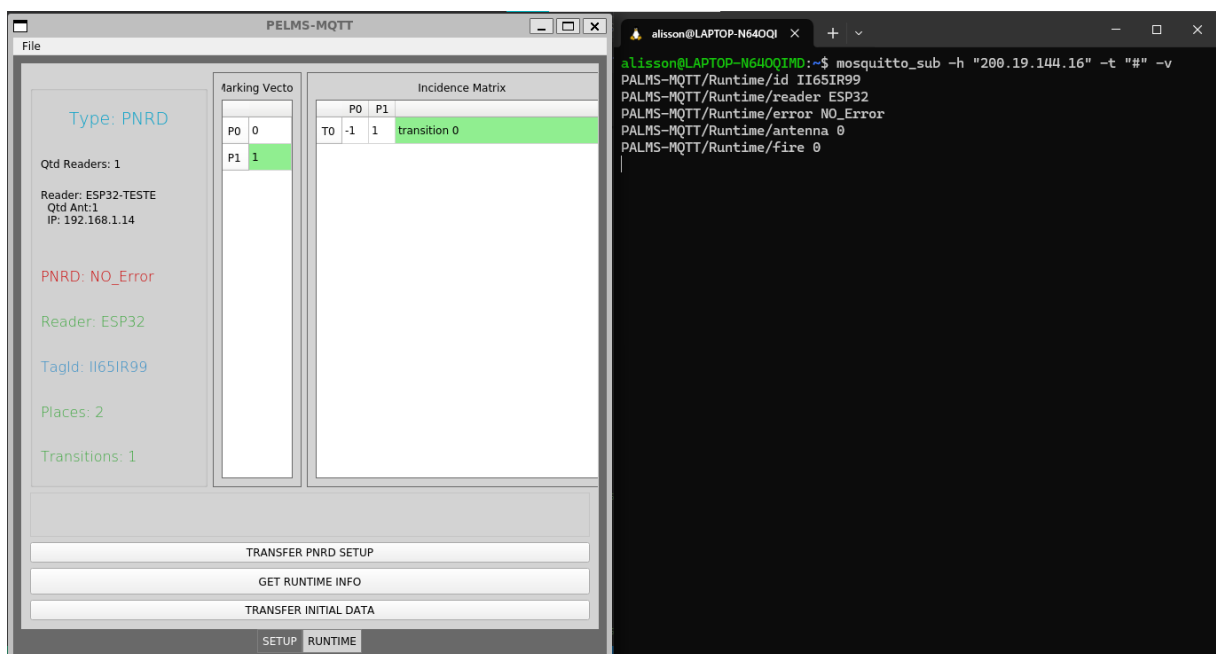


Figura 35 - Subscrição do Runtime do ESP32® Disparador.

O terceiro teste foi realizado em duas partes, onde primeiramente foram feitos os testes na plataforma ESP32 Recorder e em seguida os testes na plataforma ESP32 Reader.

Na primeira parte foi feita a compilação do código gerado pelo middleware PELMS-MQTT na plataforma ESP32® utilizando o software Arduino IDE, o código é responsável por conectar a plataforma ESP32® ao WiFi e ao Broker, subscrever os tópicos “PELMS-MQTT/len_place”, “PELMS-MQTT/len_transition”, “PELMS-MQTT/incidence_matrix”, “PELMS-MQTT/marking_vector” e “PELMS-MQTT/ fire_vector”, e gravar as variáveis de Rede de Petri nas etiquetas.

Na segunda parte foi feita a compilação do código gerado pelo middleware PELMS-MQTT na plataforma ESP32® utilizando o software Arduino IDE, o código é responsável por conectar a plataforma ESP32® ao WiFi e ao Broker, subscrever os tópicos “PELMS-MQTT/len_place”, “PELMS-MQTT/len_transition”, “PELMS-MQTT/incidence_matrix” e “PELMS-MQTT/marking_vector”, ler as etiquetas realizando o disparando da transição calculando o novo vetor de marcação, e enviar o status do disparo publicando os dados nos tópicos com o rótulo iniciado por “PELMS-MQTT/Runtime/”.

3.5 Geração dos arquivos PNML via WoPeD

Após realizados os testes, foram definidos dois modelos de Rede de Petri para este trabalho, onde representam o funcionamento tanto da PNRD, quanto da iPNRD utilizando o middleware PELMS-MQTT. Esses modelos representam a planta de teste de qualidade e retrabalho de mancais em um ambiente fabril (Figura 36) e o mapa das posições que o robô pode ocupar em uma sala (Figura 37). Com auxílio da modelagem foi utilizado a ferramenta WoPeD, ferramenta capaz de modelagem e simulação de Redes de Petri.

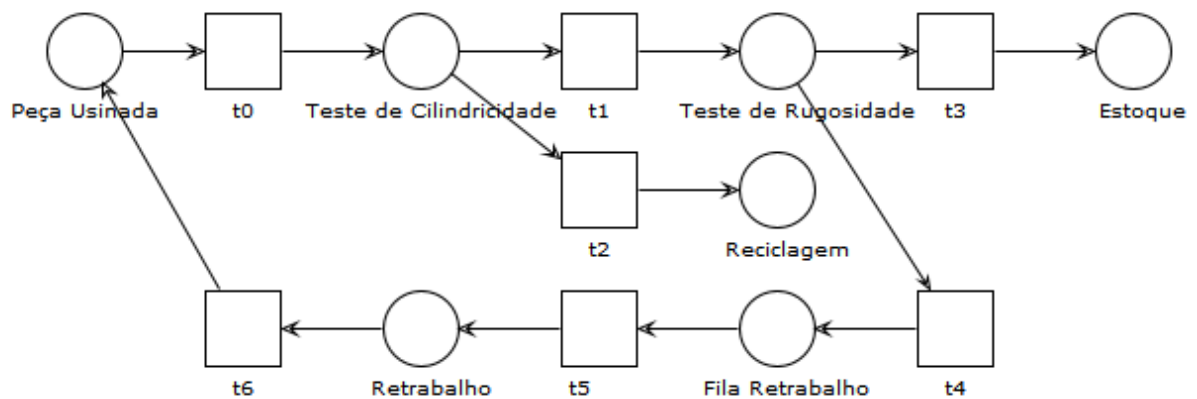


Figura 36 - Modelo de Rede de Petri de uma planta de teste de qualidade e retrabalho de mancais em um ambiente fabril.

O primeiro modelo representa a planta de teste de qualidade e retrabalho de mancais de uma fábrica (Figura 36), inicialmente entra uma peça usinada na planta onde ela percorre a cadeia de inspeção de qualidade e de retrabalho, e termina como um produto acabado ou pronto para o descarte/reciclagem. O processo é realizado utilizando três máquinas:

- A primeira máquina realiza a inspeção de cilindricidade e a leitura de duas etiquetas, a primeira leitura indica que a peça entrou na máquina e que está pronta para o teste de cilindricidade, após este teste a segunda leitura direciona a peça para reciclagem caso seja reprovado;
- A segunda máquina realiza a inspeção de rugosidade e a leitura de três etiquetas, a primeira leitura indica que a peça entrou na máquina e que está pronta para o teste de rugosidade, após o teste a segunda e a terceira leitura direciona a peça para estoque (peça aprovada) ou para fila de retrabalho (peça reprovada), respectivamente;
- A terceira máquina realiza o retrabalho e a leitura de duas etiquetas, a primeira leitura indica que a peça entrou na máquina e que está pronta para retrabalho, após o retrabalho a segunda leitura indica que a peça está acabada e pronta para um novo teste.

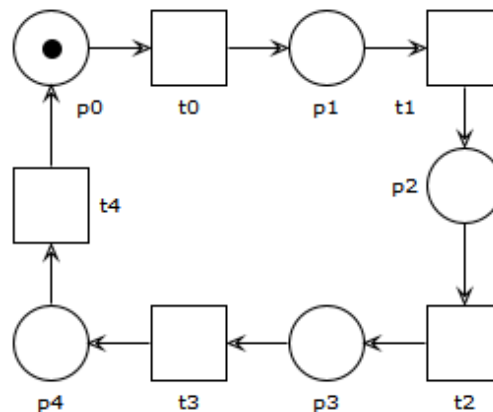


Figura 37 - Modelo de Rede de Petri de um mapa das posições que um robô pode ocupar em uma sala.

O segundo modelo representa o mapa das posições que o robô pode ocupar em uma sala (Figura 37), inicialmente o robô ocupa a posição “p0” e percorre todo o trajeto passando pelos pontos de controle, retornando para a posição inicial. O trajeto em regime de funcionamento possui cinco pontos de controle:

- O primeiro ponto de controle é o a posição “p0”, nele encontra-se um leitor que dispara a transição “t4” mudando o status de posição “p4” para “p0”;
- O segundo ponto de controle é o a posição “p1”, nele encontra-se um leitor que dispara a transição “t0” mudando o status de posição “p0” para “p1”;

- O terceiro ponto de controle é o a posição “*p2*”, nele encontra-se um leitor que dispara a transição “*t1*” mudando o status de posição “*p1*” para “*p2*”;
- O quarto ponto de controle é o a posição “*p3*”, nele encontra-se um leitor que dispara a transição “*t2*” mudando o status de posição “*p2*” para “*p3*”;
- O quinto ponto de controle é o a posição “*p4*”, nele encontra-se um leitor que dispara a transição “*t3*” mudando o status de posição “*p3*” para “*p4*”, e assim por diante;

Caso o robô pule algum ponto de controle ou algum ponto de controle pare de funcionar os demais pontos de controle são responsáveis por identificar o evento e dispara uma exceção.

3.6 PELMS-MQTT

O PELMS-MQTT possui dois módulos “SETUP” e “RUNTIME”, o primeiro módulo interpreta e converte o arquivo PNML em variáveis da Rede de Petri, e as armazena juntamente com o tipo de variante da PNRD e os dados das plataformas em arquivos “*setup.pelms*”, “*pnrdData.pnrd*”, “*pnrd*.ino*” e “*WiFiPassword.h*” (Figura 38 e Figura 39 – Módulo Setup), o segundo módulo realiza a leitura do arquivo “*pnrdData.pnrd*” e envia os dados da RP para as plataformas de gravação e disparo da Rede de Petri (Figura 33, Figura 34 e Figura 39 – Módulo *Runtime*: Software PELMS), após os dados serem enviados, o middleware aguarda o retorno do status do disparo via MQTT e o armazena em um arquivo “*runtime.pnrd*” (Figura 35 e Figura 39 -Módulo *Runtime*: Mudança de Estado do Modelo).

```

alisson@LAPTOP-N64QQIMD:/mnt/c/Users/engal/Downloads/PFC/PELMS_Simulation/PELMS-MQTT-main$ ./main.py
Diretório /mnt/c/Users/engal/Downloads/Bearing_Quality_Testing criado com sucesso
Arquivo Bearing_Quality_Testing.pnml copiado com sucesso para /mnt/c/Users/engal/Downloads/Bearing_Quality_Testing/Bearing_Quality_Testing.pnml
Diretório /mnt/c/Users/engal/Downloads/Bearing_Quality_Testing/pelmsSetup criado com sucesso
Arquivo /mnt/c/Users/engal/Downloads/Bearing_Quality_Testing/pelmsSetup/setup.pelms criado com sucesso
Arquivo /mnt/c/Users/engal/Downloads/Bearing_Quality_Testing/pelmsSetup/pnrdData.pnrd criado com sucesso
Diretório /mnt/c/Users/engal/Downloads/Bearing_Quality_Testing/pelmsSetup/ino/pnrd_recorder criado com sucesso
Arquivo /mnt/c/Users/engal/Downloads/Bearing_Quality_Testing/pelmsSetup/ino/pnrd_recorder/pnrd_recorder.ino criado com sucesso
Arquivo /mnt/c/Users/engal/Downloads/Bearing_Quality_Testing/pelmsSetup/ino/pnrd_recorder/WiFiPassword.h criado com sucesso
Diretório /mnt/c/Users/engal/Downloads/Bearing_Quality_Testing/pelmsSetup/ino/pnrd_reader_0 criado com sucesso
Arquivo /mnt/c/Users/engal/Downloads/Bearing_Quality_Testing/pelmsSetup/ino/pnrd_reader_0/pnrd_reader_0.ino criado com sucesso
Arquivo /mnt/c/Users/engal/Downloads/Bearing_Quality_Testing/pelmsSetup/ino/pnrd_reader_0/WiFiPassword.h criado com sucesso
Diretório /mnt/c/Users/engal/Downloads/Bearing_Quality_Testing/pelmsSetup/ino/pnrd_reader_1 criado com sucesso
Arquivo /mnt/c/Users/engal/Downloads/Bearing_Quality_Testing/pelmsSetup/ino/pnrd_reader_1/pnrd_reader_1.ino criado com sucesso
Arquivo /mnt/c/Users/engal/Downloads/Bearing_Quality_Testing/pelmsSetup/ino/pnrd_reader_1/WiFiPassword.h criado com sucesso
Diretório /mnt/c/Users/engal/Downloads/Bearing_Quality_Testing/pelmsSetup/ino/pnrd_reader_2 criado com sucesso
Arquivo /mnt/c/Users/engal/Downloads/Bearing_Quality_Testing/pelmsSetup/ino/pnrd_reader_2/pnrd_reader_2.ino criado com sucesso
Arquivo /mnt/c/Users/engal/Downloads/Bearing_Quality_Testing/pelmsSetup/ino/pnrd_reader_2/WiFiPassword.h criado com sucesso
Diretório /mnt/c/Users/engal/Downloads/Robot_Positioning criado com sucesso
Arquivo Robot_Positioning.pnml copiado com sucesso para /mnt/c/Users/engal/Downloads/Robot_Positioning/Robot_Positioning.pnml
Diretório /mnt/c/Users/engal/Downloads/Robot_Positioning/pelmsSetup criado com sucesso
Arquivo /mnt/c/Users/engal/Downloads/Robot_Positioning/pelmsSetup/setup.pelms criado com sucesso
Arquivo /mnt/c/Users/engal/Downloads/Robot_Positioning/pelmsSetup/ipnrdData.pnrd criado com sucesso
Diretório /mnt/c/Users/engal/Downloads/Robot_Positioning/pelmsSetup/ino/ipnrd_recorder criado com sucesso
Arquivo /mnt/c/Users/engal/Downloads/Robot_Positioning/pelmsSetup/ino/ipnrd_recorder/ipnrd_recorder.ino criado com sucesso
Arquivo /mnt/c/Users/engal/Downloads/Robot_Positioning/pelmsSetup/ino/ipnrd_recorder/WiFiPassword.h criado com sucesso
Diretório /mnt/c/Users/engal/Downloads/Robot_Positioning/pelmsSetup/ino/ipnrd_reader_0 criado com sucesso
Arquivo /mnt/c/Users/engal/Downloads/Robot_Positioning/pelmsSetup/ino/ipnrd_reader_0/ipnrd_reader_0.ino criado com sucesso
Arquivo /mnt/c/Users/engal/Downloads/Robot_Positioning/pelmsSetup/ino/ipnrd_reader_0/WiFiPassword.h criado com sucesso
Diretório /mnt/c/Users/engal/Downloads/Robot_Positioning/pelmsSetup/ino/ipnrd_reader_1 criado com sucesso
Arquivo /mnt/c/Users/engal/Downloads/Robot_Positioning/pelmsSetup/ino/ipnrd_reader_1/ipnrd_reader_1.ino criado com sucesso
Arquivo /mnt/c/Users/engal/Downloads/Robot_Positioning/pelmsSetup/ino/ipnrd_reader_1/WiFiPassword.h criado com sucesso
Diretório /mnt/c/Users/engal/Downloads/Robot_Positioning/pelmsSetup/ino/ipnrd_reader_2 criado com sucesso
Arquivo /mnt/c/Users/engal/Downloads/Robot_Positioning/pelmsSetup/ino/ipnrd_reader_2/ipnrd_reader_2.ino criado com sucesso
Arquivo /mnt/c/Users/engal/Downloads/Robot_Positioning/pelmsSetup/ino/ipnrd_reader_2/WiFiPassword.h criado com sucesso
Diretório /mnt/c/Users/engal/Downloads/Robot_Positioning/pelmsSetup/ino/ipnrd_reader_3 criado com sucesso
Arquivo /mnt/c/Users/engal/Downloads/Robot_Positioning/pelmsSetup/ino/ipnrd_reader_3/ipnrd_reader_3.ino criado com sucesso
Arquivo /mnt/c/Users/engal/Downloads/Robot_Positioning/pelmsSetup/ino/ipnrd_reader_3/WiFiPassword.h criado com sucesso
Diretório /mnt/c/Users/engal/Downloads/Robot_Positioning/pelmsSetup/ino/ipnrd_reader_4 criado com sucesso
Arquivo /mnt/c/Users/engal/Downloads/Robot_Positioning/pelmsSetup/ino/ipnrd_reader_4/ipnrd_reader_4.ino criado com sucesso
Arquivo /mnt/c/Users/engal/Downloads/Robot_Positioning/pelmsSetup/ino/ipnrd_reader_4/WiFiPassword.h criado com sucesso

```

Figura 38 - Arquivos gerados pelo middleware PELMS-MQTT.

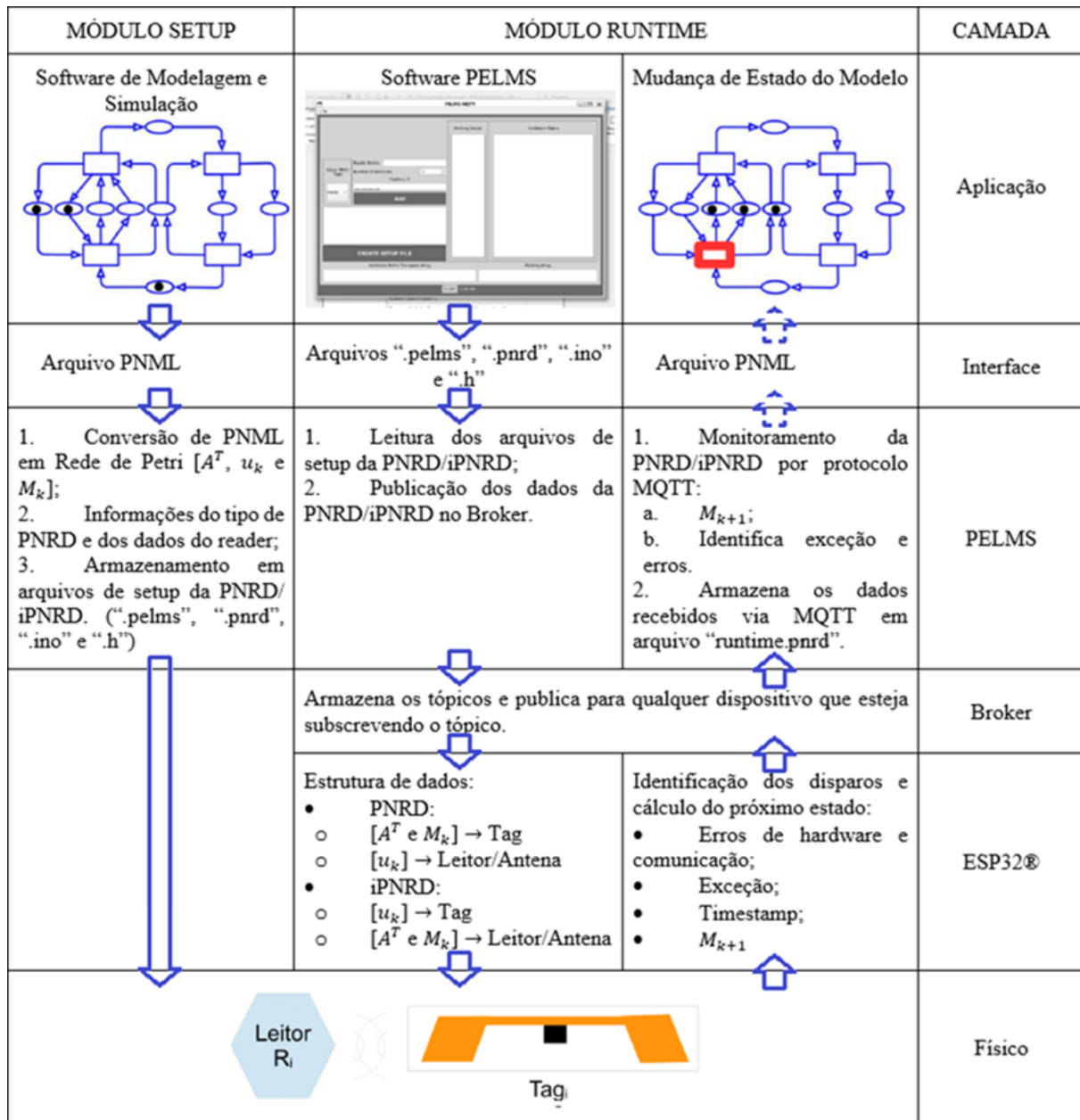


Figura 39 – Esquema de funcionamento do middleware PELMS-MQTT.

3.6.1 Implementações/Melhorias PELMS-MQTT

Durante o processo de adaptação do middleware PALMS-MQTT para PELMS-MQTT foram feitas alterações tanto no *front-end* quanto no *back-end*. O *front-end* do *middleware* é apresentado pelo arquivo “*MainWindow.py*” (APÊNDICE C) onde ele é responsável por criar o menu contendo botões para abrir arquivos PNML e PELMS, e o botão para sair (Figura 40), ele cria abas “*SETUP*” e “*RUNTIME*”. A aba “*SETUP*”, possui lista do tipo de PNRD (*Setup PNRD Type*), caixas de listagem das entradas de dados (*Reader Name*, *Number of Antennas* e *Platform IP*) e das saídas de dados (*Create Setup File Menu*, *Marking Vector*, *Incidence Matrix*, *Incidence Matrix Transpose Array* e *Marking Array*), e botões para adicionar os dados de

entrada (*Add*) e para criar os arquivos para o funcionamento do *middleware* PELMS-MQTT (Figura 40).

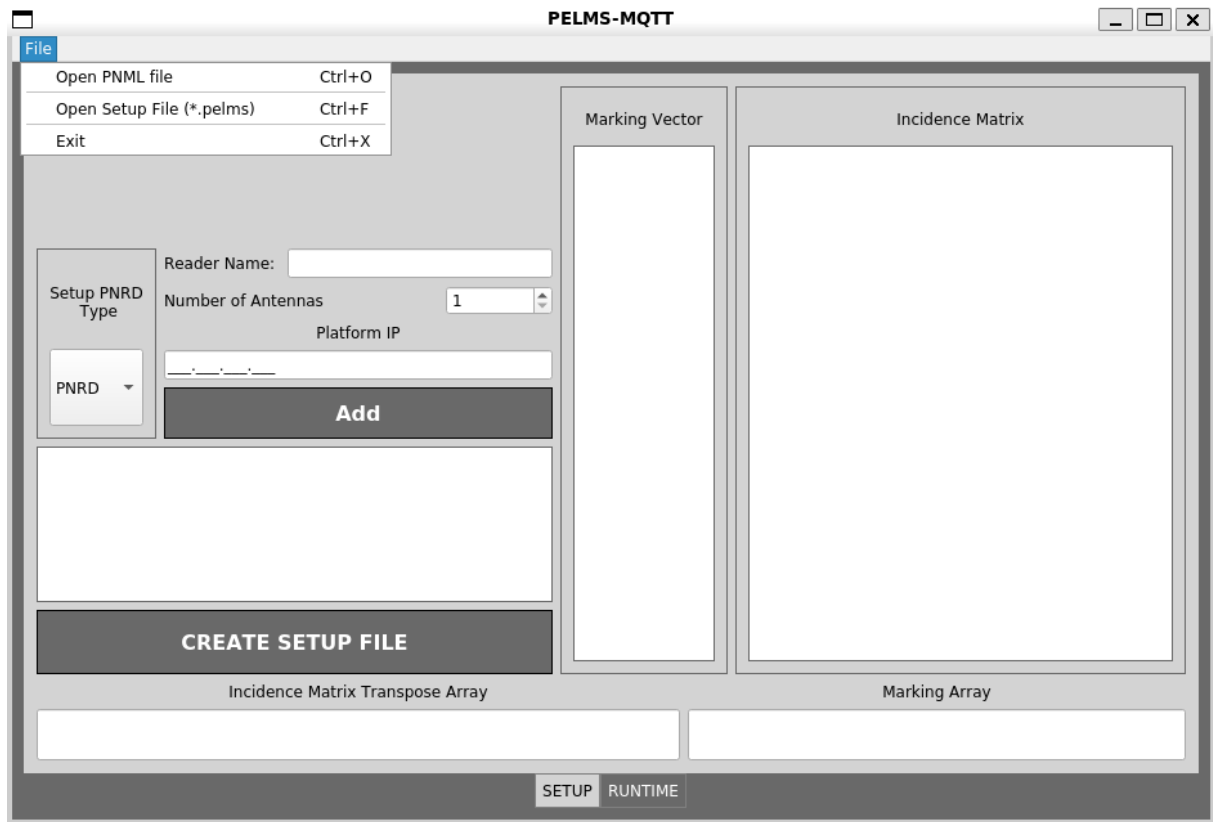


Figura 40 - Menu e aba "SETUP" do middleware PELMS-MQTT.

A aba "*RUNTIME*", possui caixas de listagem das saídas de dados (Dados da PNRD e dos dados do disparo, *Marking Vector*, *Incidence Matrix*), *label* dos dados de funcionamento da Rede de Petri, e botões para transferir dados para o disparo da Rede de Petri (*TRANSFER PNRD SETUP*), para inicialização da RP (*TRANSFER INITIAL DATA*) e para receber dados dos disparos da RP (*GET RUNTIME INFO*), que libera a comunicação MQTT (Figura 41).

O *back-end* do middleware é apresentado pelos demais arquivos ".py", ".ino" e ".h", os arquivos "*Place.py*", "*Arc.py*", "*Transition.py*", "*PetriNet.py*", "*Pnrd.py*" e "*pnml.py*" são arquivos de interpretação e conversão de arquivos PNML em variáveis de Rede de Petri, e os arquivos "*FileCopy.py*" e "*FileCreator.py*" são arquivos para cópia e de criação de arquivos, já os arquivos "*main.py*" e "*PelmsGui.py*" (APÊNDICE D) são os arquivos principais que manipulam o funcionamento dos botões dentro do menu e das janelas "*SETUP*" e "*RUNTIME*", a janelas contém as caixas de entrada e de saída, e os botões de adicionar os dados de entrada, de criar os arquivos, de transferência de dados para o disparo e para inicialização da RP, e de liberação da comunicação MQTT (Figura 40 e Figura 41).

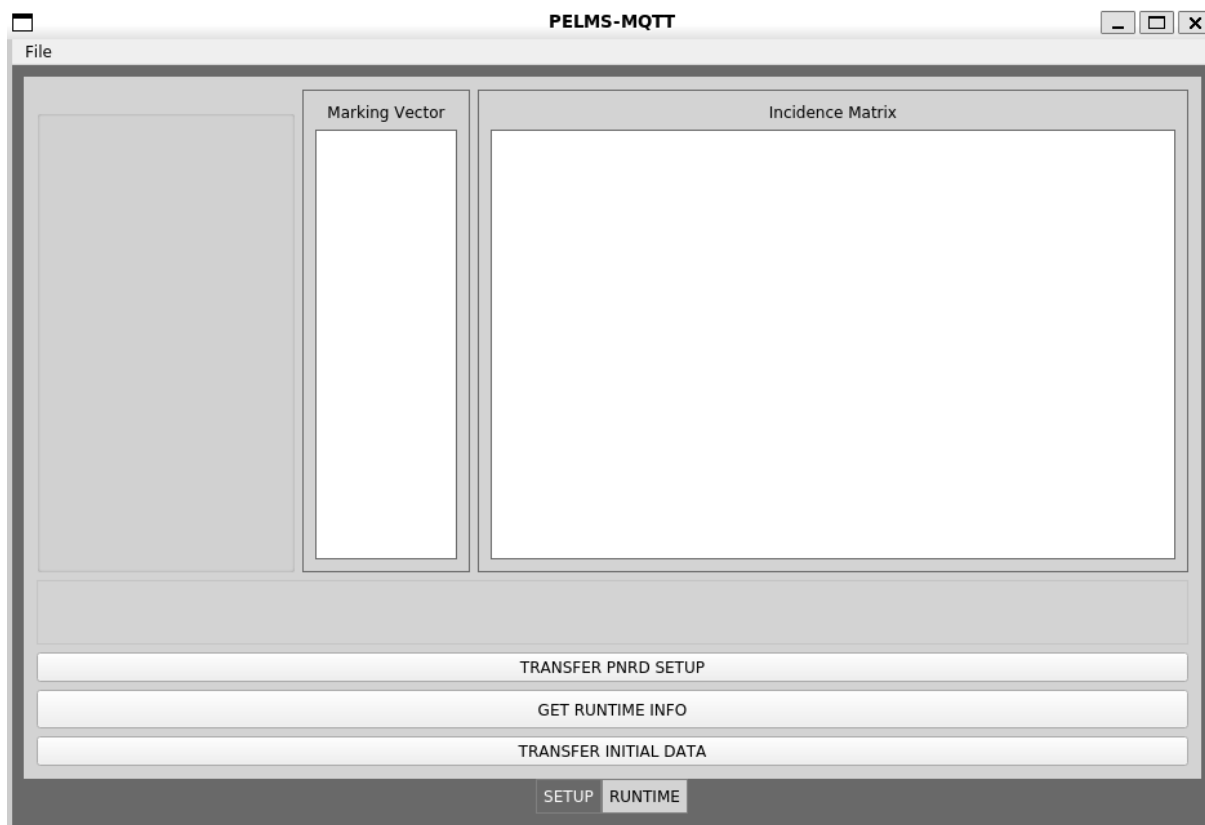


Figura 41 – Aba “RUNTIME” do middleware.

Por fim, existem os arquivos de template, onde o código do arquivo “*template.py*” (APÊNDICE E) é responsável por criar os textos que serão inseridos nos arquivos da Figura 38, nele são gerados o texto dos arquivos “*runtime.py*” e “**pnrdData.pnrd*”, e são lidos os arquivos “*template_begin.ino*” (APÊNDICE F), “*template_reader_*pnrd.ino*” (APÊNDICE G e APÊNDICE H), “*template_recorder_*pnrd.ino*” (APÊNDICE I e APÊNDICE J) e “*template_wifipassword.h*” (APÊNDICE K) para geração dos arquivos de carregamento que serão utilizados na ferramenta Arduino IDE, para serem compilados e carregados nos ESP32, nesses arquivos existem as palavras chaves que serão substituídas pelas variáveis da comunicação MQTT e as variáveis dos leitores.

3.7 Montagem das placas ESP32® Recorder e Reader

Nesse projeto foram utilizados como componentes para os resultados do funcionamento do PELMS-MQTT, duas plataformas ESP32, dois leitores PN532-NFC-Reader e dois LCDs com dimensões 16x2. Por meio da ferramenta *Fritzing* foi possível montar um esquema de ligação desses componentes (Figura 42). Finalizado o esquema, foi feita a montagem real dos dispositivos ESP32-Recorder (Figura 43) e ESP32-Reader (Figura 43) que serão utilizados.

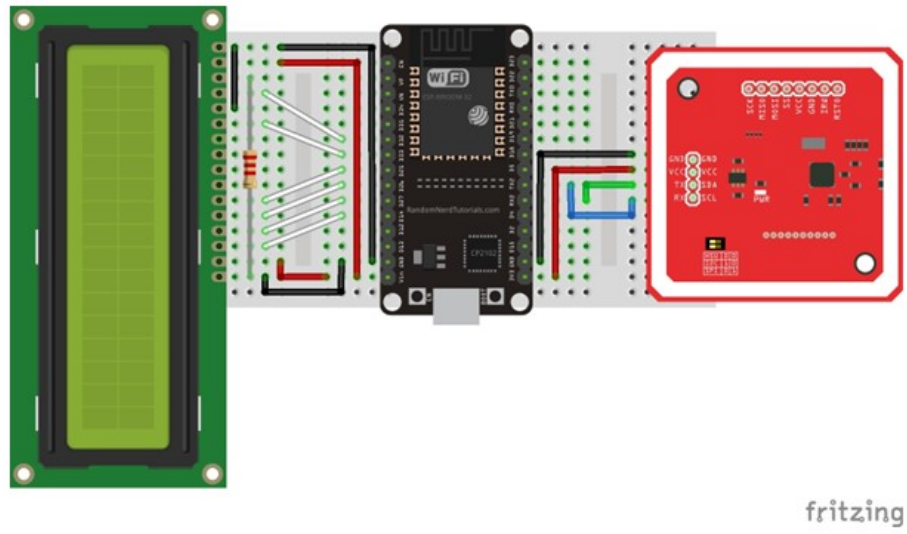


Figura 42 - Esquema de Ligação do ESP32.

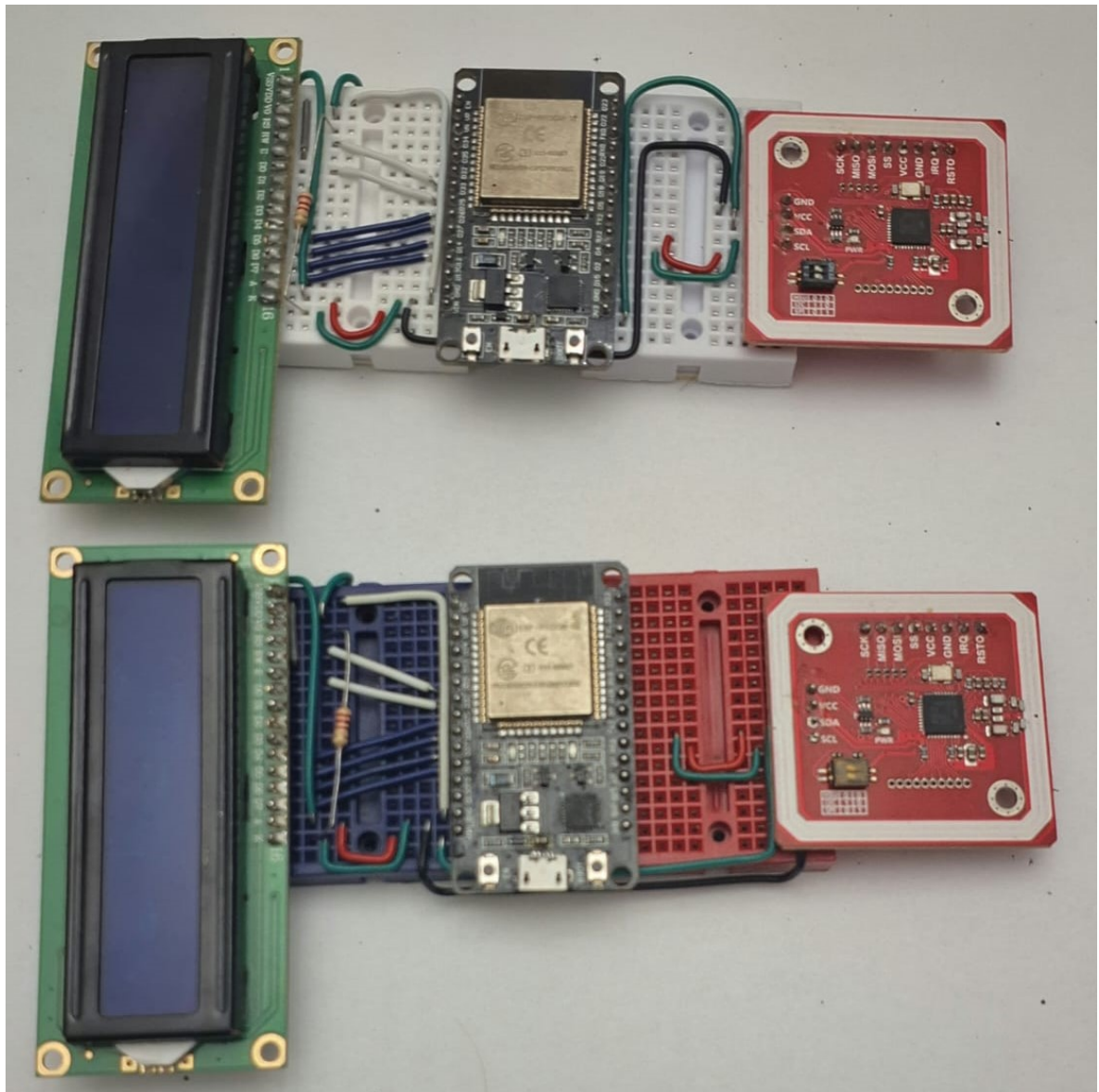


Figura 43 - Montagem do ESP32 Recorder e Reader.

4 RESULTADOS

Nesse capítulo, serão abordados os resultados da implementação do PELMS-MQTT para os modelos de Rede de Petri de uma planta de teste de qualidade e retrabalho de mancais em um ambiente fabril e de um mapa das posições que o robô pode ocupar em uma sala. Os códigos utilizados para obter os resultados do funcionamento do middleware PELMS-MQTT estão disponíveis em <<https://github.com/MAPL-UFU/PELMS>>.

Os módulos *SETUP* e *RUNTIME* abaixo representa o passo a passo para a obtenção dos resultados, desde a geração dos arquivos até a simulação.

4.1 Módulo *SETUP*

No módulo *SETUP* as Redes de Petri são carregadas por meio da interpretação/conversão dos arquivos PNML “*Bearing_Quality_Testing.pnml*” e “*Robot_Positioning.pnml*”, obtendo a Matriz de Incidência e o Vetor de Marcações. As Figura 44 e Figura 45 exemplificam como é carregado a Rede de Petri no middleware PELMS-MQTT.

Com a Rede de Petri carregada (Figura 48 e Figura 59), deve-se selecionar o tipo de PNRD e inserir os dados do *reader*, como nome do *reader*, quantidade de antena e o IP de cada ESP32®. Inseridos todos os *reader* o usuário pode pressionar o botão “*CREATE SETUP FILE*” (Figura 48 e Figura 59) para gerar o arquivo “*setup.pelms*” do middleware PELMS-MQTT, gerado o arquivo será apresentado um aviso de criação de todos os arquivos para o middleware PELMS-MQTT e a ferramenta Arduino IDE(Figura 46).

4.2 Módulo *RUNTIME*

O módulo *RUNTIME* é abrindo o arquivo “*setup.pelms*” no menu do middleware, contendo os caminhos dos arquivos PNML “*Bearing_Quality_Testing.pnml*” e “*Robot_Positioning.pnml*”, o *middleware* interpreta/converte os dados da PNML em dados de Rede de Petri. As Figura 44 e Figura 47 exemplificam como é aberto o arquivo “*setup.pelms*” no *middleware* PELMS-MQTT.

Com a abertura do arquivo “*setup.pelms*” o módulo *RUNTIME* é apresentado com os dados do tipo de PNRD, do *reader*, da dimensão da Rede de Petri, da Matriz de Incidência e o Vetor de Marcações (Figura 49 e Figura 60). Pressionando o botão “*TRANSFER INITIAL PNRD*”, os dados de gravação da PNRD são enviados via MQTT para o ESP32-Recorder que irá salvar os dados na Tag. As Figura 50 e Figura 61 mostram o ESP32® recebendo os dados

da PNRD enviados pelo middleware PELMS-MQTT. Após receber as mensagens, as Figura 51, Figura 52, Figura 62 e Figura 63 mostram o ESP32® salvando os dados da PNRD na Tag.

Pressionando o botão “*TRANSFER PNRD SETUP*”, os dados do disparo da PNRD são enviados via MQTT para o ESP32-Reader que irá disparar a PNRD, lendo os dados contidos na Tag e calculando um novo vetor de marcação. As Figura 53 e Figura 64 mostram o ESP32® recebendo os dados da PNRD enviados pelo middleware PELMS-MQTT. Após receber as mensagens, as Figura 54, Figura 55, Figura 65 e Figura 66 mostram o ESP32® disparando e o salvando os dados da PNRD na Tag. Em seguida nas Figura 56 e Figura 67 mostram o envio dos dados do disparo para o *middleware*.

Pressionando o botão “*GET RUNTIME INFO*”, é liberado a subscrição de todos os tópicos com rótulo iniciado por “*PELMS-MQTT/Runtime/*”, o middleware ao receber um status do disparo apresenta na “*label*” os dados do disparo, contendo informações do ID da Tag, o status do disparo e o ID do leitor, armazenando os dados em um arquivo “*runtime.pnr*”. As Figura 57 e Figura 68 mostram o PELMS-MQTT recebendo um disparo NO_Error e as Figura 58 e Figura 69 mostram o PELMS-MQTT recebendo um disparo Exception.

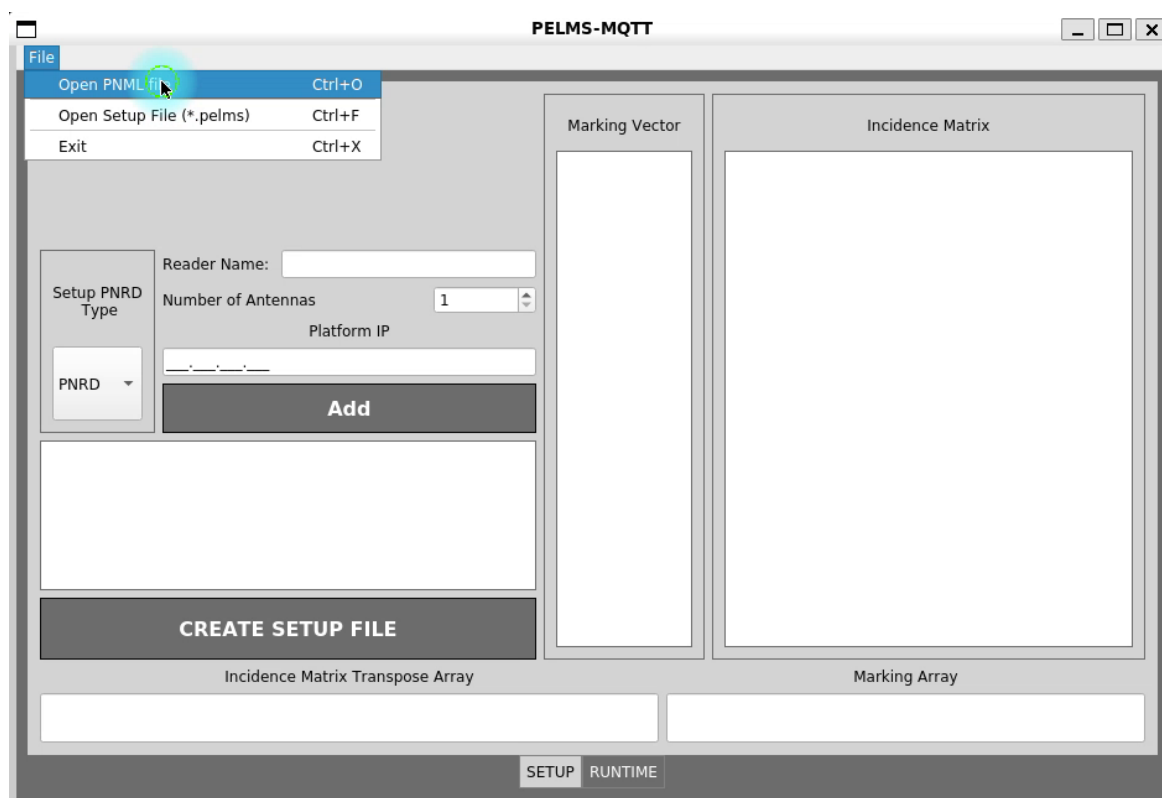


Figura 44 - Menu de opções no software PELMS-MQTT.

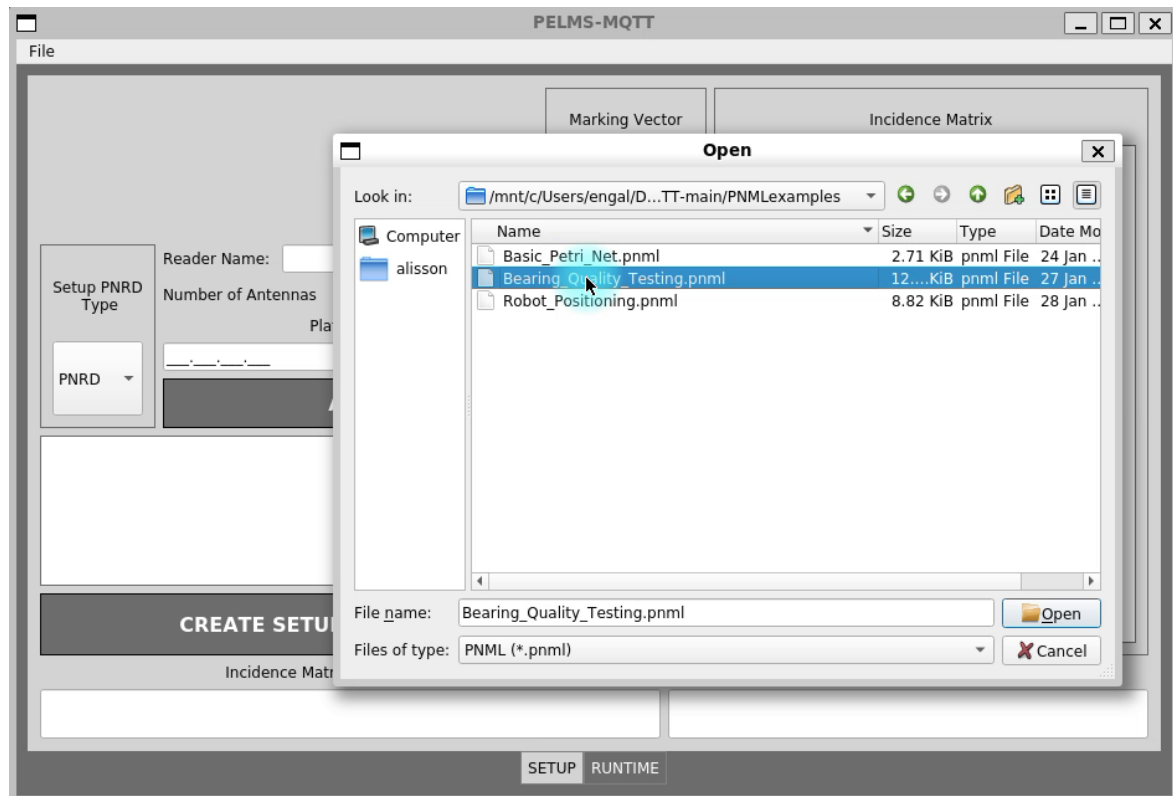


Figura 45 - Janela de abertura de arquivo PNML no software PELMS-MQTT

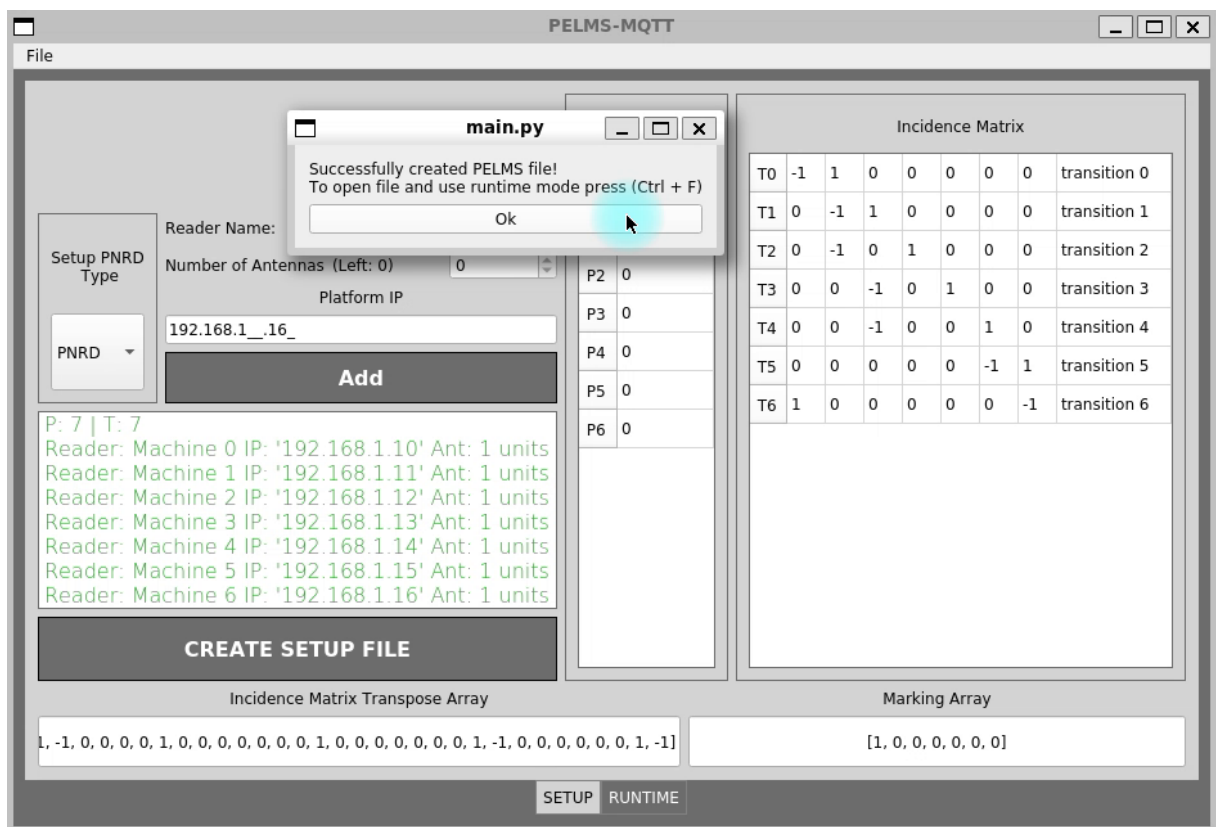


Figura 46 - Popup gerado com a informação de arquivos gerados.

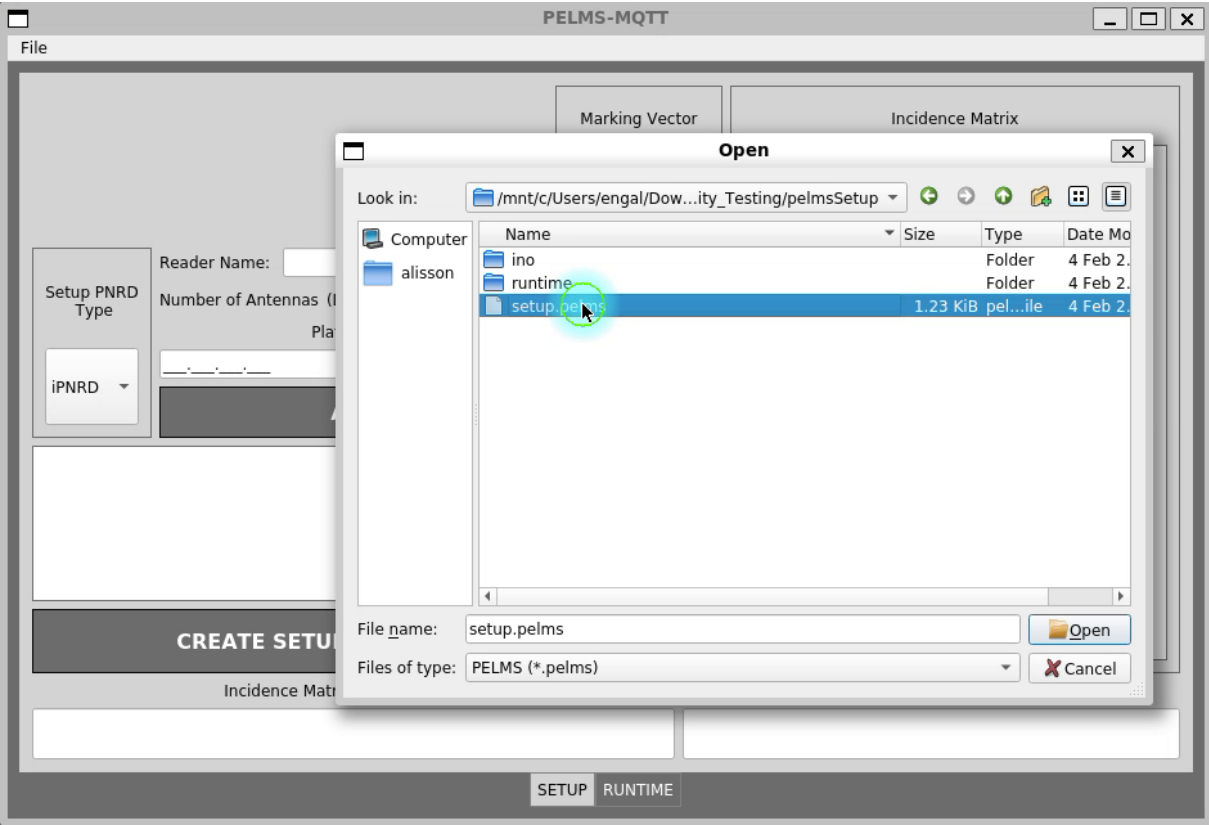


Figura 47 - Janela de abertura de arquivo PELMS no software PELMS-MQTT.

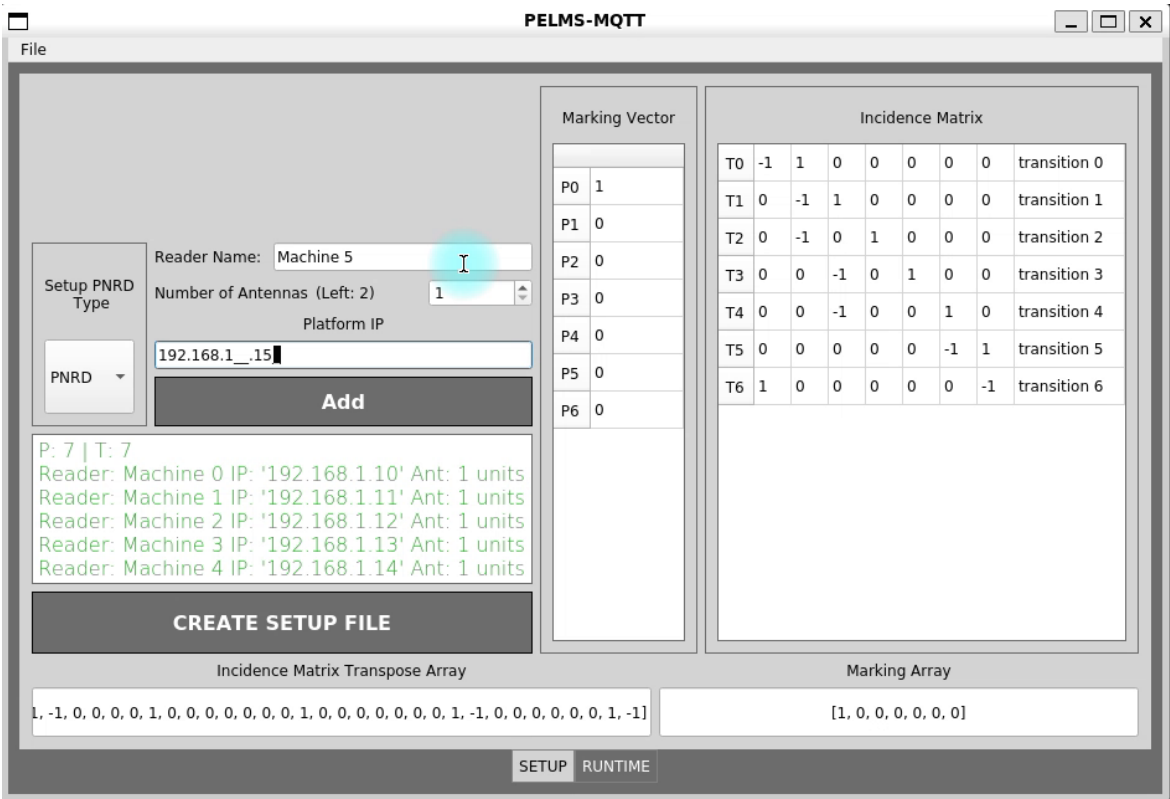


Figura 48 - Rede de Petri e dados dos reader gerados no middleware PELMS-MQTT.

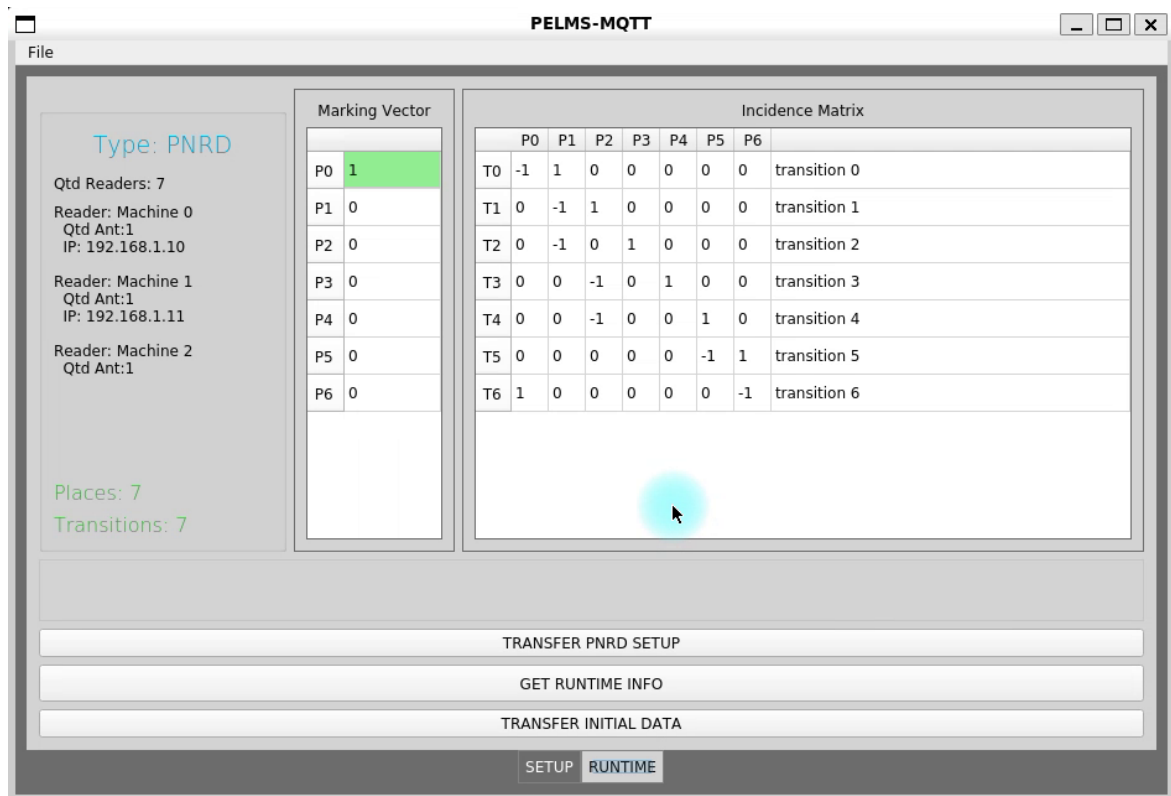


Figura 49 - Aba RUNTIME no middleware PELMS-MQTT.

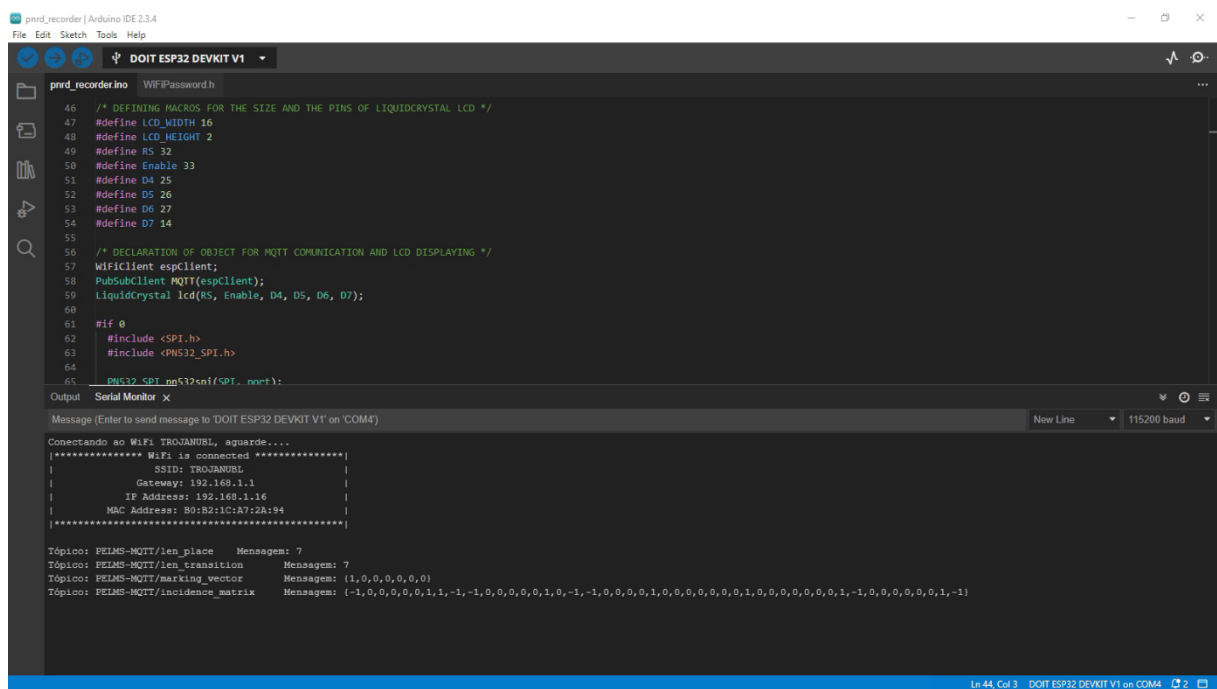
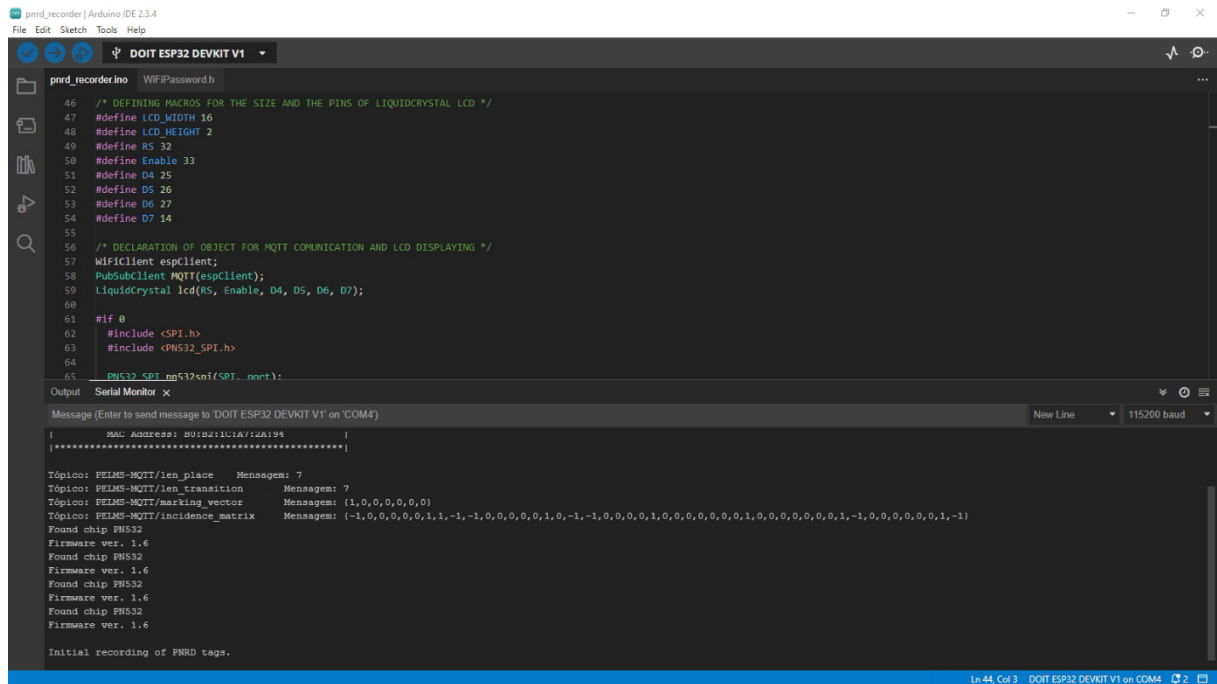


Figura 50 - Recebimento de dados via MQTT enviadas pelo middleware PELMS-MQTT para o ESP32-Recorder.



The screenshot shows the Arduino IDE interface with the file 'pnrd_recorder.ino' open. The code defines macros for an LCD and includes libraries for SPI and PN532. The Serial Monitor is open, displaying the following output:

```

MAC Address: B8:27:1C:A7:DA:96
-----
Tópico: FEIIMS-MQTT/len_place      Mensagem: 7
Tópico: FEIIMS-MQTT/len_transition Mensagem: 7
Tópico: FEIIMS-MQTT/matrix_vector Mensagem: (1,0,0,0,0,0,0)
Tópico: FEIIMS-MQTT/incidence_matrix Mensagem: (-1,0,0,0,0,1,1,-1,-1,0,0,0,0,1,0,-1,-1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,1,-1,0,0,0,0,0,1,-1)
Found chip PN532
Firmware ver. 1.6
Found chip PN532
Firmware ver. 1.6
Found chip PN532
Firmware ver. 1.6
Found chip PN532
Firmware ver. 1.6
Initial recording of PNRD tags.

```

Figura 51 - Salvamento dos dados da PNRD na Tag no Serial Monitor para o ESP32-Recorder.

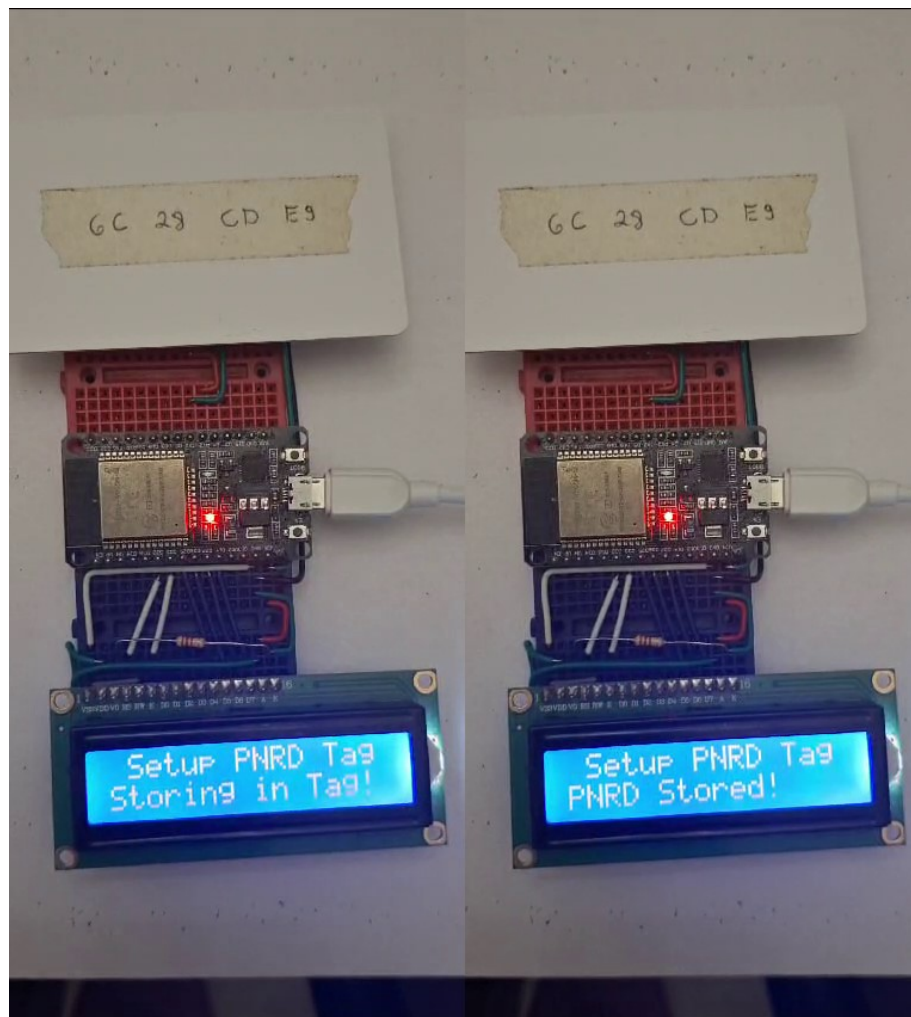


Figura 52 - Salvamento dos dados da PNRD na Tag apresentado na tela LCD 16x2.

The screenshot displays the Arduino IDE interface. The top menu bar includes 'File', 'Edit', 'Sketch', 'Tools', and 'Help'. The toolbar shows icons for opening files, saving, compiling, and uploading. The main editor window is titled 'prv_read_0.ino' and contains the following C++ code:

```
163
164 printf(start ? "Tópico: %s\tMensagem: %s\n":"Subscribe:\nTópico: %s\tMensagem: %s\n", msg_topic.c_str(), msg_c_str());
165
166 if (status != msg || topic_status != msg_topic) {
167     status = msg;
168     topic_status = msg_topic;
169     msg.toLowerCase();
170     if (!strcmp(msg_topic.c_str(), "PELMS-MQTT/fire/len_place")) {
171         sizearray[0] = msg.toInt();
172         start = true;
173     } else if (!strcmp(msg_topic.c_str(), "PELMS-MQTT/fire/len_transition") && start) {
174         sizearray[1] = msg.toInt();
175
176         // printf("Places: %d\nTransition: %d\nReader: %s\nAntenna: %s\nFire Position: %d\n", sizearray[0], sizearray[1], pelmsReader.c_str(), pelmsAntenna.c_str(), pelmsPosition.c_str());
177         LCD_print(1, "Starting", LCD_WIDTH + 1, "left", 700);
178
179         start_PNRD();
180         start = false;
181     }
182 }
```

The 'Serial Monitor' window is open at the bottom, showing the output of the program. It is set to 'New Line' and '115200 baud'. The output text is as follows:

```
.....
Conexões MQTT rodando no nucleo 0
Subscribe:
Tópico: PELMS-MQTT/fire/len_place      Mensagem: 7
Tópico: PELMS-MQTT/fire/len_transition Mensagem: 7
Found chip BMS32
Firmware ver. 1.6
Token Vector:
0
1
0
0
0
0
0
0
```

[illegible]

Figura 56 - Envio dos dados via MQTT do ESP32-Reader para o middleware PELMS-MQTT referente ao disparo.

File

PELMS-MQTT

Type: PNRD

Qtd Readers: 7

Reader: Machine 0
Qtd Ant:1
IP: 192.168.1.10

Reader: Machine 1
Qtd Ant:1
IP: 192.168.1.11

Reader: Machine 2
Qtd Ant:1

PNRD: NO_Error

Reader: Machine 0

TagId: A405111E

Places: 7

Transitions: 7

P0	0
P1	1
P2	0
P3	0
P4	0
P5	0
P6	0

	P0	P1	P2	P3	P4	P5	P6	
T0	-1	1	0	0	0	0	0	transition 0
T1	0	-1	1	0	0	0	0	transition 1
T2	0	-1	0	1	0	0	0	transition 2
T3	0	0	-1	0	1	0	0	transition 3
T4	0	0	-1	0	0	1	0	transition 4
T5	0	0	0	0	0	-1	1	transition 5
T6	1	0	0	0	0	0	-1	transition 6

TRANSFER PNRD SETUP

GET RUNTIME INFO

TRANSFER INITIAL DATA

SETUP RUNTIME

Figura 57 – Disparo de uma transição no middleware PELMS-MQTT.

File

PELMS-MQTT

Type: PNRD

Qtd Readers: 7

Reader: Machine 0
Qtd Ant:1
IP: 192.168.1.10

Reader: Machine 1
Qtd Ant:1
IP: 192.168.1.11

Reader: Machine 2
Qtd Ant:1

PNRD: Exception

Reader: Machine 0

TagId: 523955DD

Places: 7

Transitions: 7

P0	0
P1	0
P2	0
P3	1
P4	1
P5	0
P6	1

	P0	P1	P2	P3	P4	P5	P6	
T0	-1	1	0	0	0	0	0	transition 0
T1	0	-1	1	0	0	0	0	transition 1
T2	0	-1	0	1	0	0	0	transition 2
T3	0	0	-1	0	1	0	0	transition 3
T4	0	0	-1	0	0	1	0	transition 4
T5	0	0	0	0	0	-1	1	transition 5
T6	1	0	0	0	0	0	-1	transition 6

TRANSFER PNRD SETUP

GET RUNTIME INFO

TRANSFER INITIAL DATA

SETUP RUNTIME

Figura 58 – Excessão de uma Tag no middleware PELMS-MQTT.

File **PELMS-MQTT** [-] [] [X]

Setup PNRD Type

Reader Name:

Number of Antennas (Left: 0)

Platform IP

P: 5 | T: 5

Reader: Position 0 IP: '192.168.1.10' Ant: 1 unit

Reader: Position 1 IP: '192.168.1.11' Ant: 1 unit

Reader: Position 2 IP: '192.168.1.12' Ant: 1 unit

Reader: Position 3 IP: '192.168.1.13' Ant: 1 unit

Reader: Position 4 IP: '192.168.1.14' Ant: 1 unit

CREATE SETUP FILE

Marking Vector

P0	1
P1	0
P2	0
P3	0
P4	0

Incidence Matrix

T0	-1	1	0	0	0	transition 0
T1	0	-1	1	0	0	transition 1
T2	0	0	-1	1	0	transition 2
T3	0	0	0	-1	1	transition 3
T4	1	0	0	0	-1	transition 4

Incidence Matrix Transpose Array

[-1, 0, 0, 0, 1, 1, -1, 0, 0, 0, 0, 1, -1, 0, 0, 0, 0, 1, -1, 0, 0, 0, 0, 1, -1]

Marking Array

[1, 0, 0, 0, 0]

SETUP **RUNTIME**

Figura 59 - Rede de Petri e dados dos reader gerados no middleware PELMS-MQTT.

File **PELMS-MQTT** [-] [] [X]

Type: iPNRD

Qtd Readers: 5

Reader: Position 0
Qtd Ant:1
IP: 192.168.1.10

Reader: Position 1
Qtd Ant:1
IP: 192.168.1.11

Reader: Position 2
Qtd Ant:1

Places: 5

Transitions: 5

Marking Vector

P0	1
P1	0
P2	0
P3	0
P4	0

Incidence Matrix

	P0	P1	P2	P3	P4	
T0	-1	1	0	0	0	transition 0
T1	0	-1	1	0	0	transition 1
T2	0	0	-1	1	0	transition 2
T3	0	0	0	-1	1	transition 3
T4	1	0	0	0	-1	transition 4

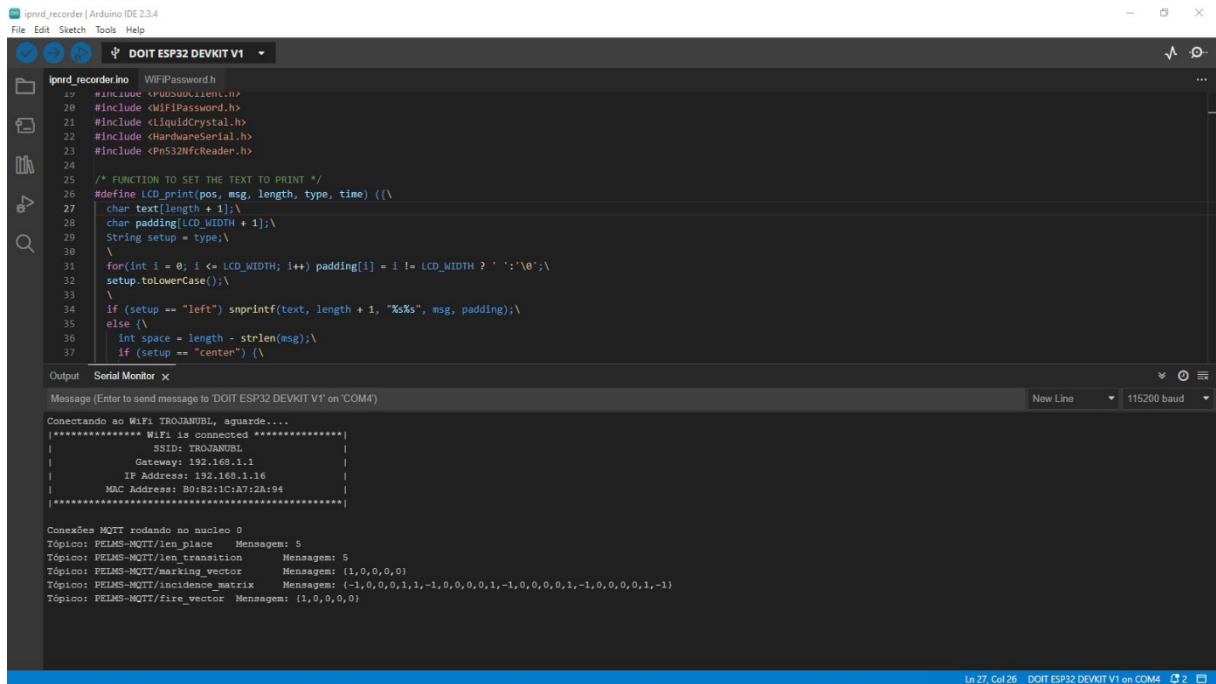
TRANSFER PNRD SETUP

GET RUNTIME INFO

TRANSFER INITIAL DATA

SETUP **RUNTIME**

Figura 60 - Aba RUNTIME no middleware PELMS-MQTT.



```

ipnrd_recorder.ino  WiFiPassword.h
19 #include <PubSubClient.h>
20 #include <WiFiPassword.h>
21 #include <LiquidCrystal.h>
22 #include <HardwareSerial.h>
23 #include <Pn532NfcReader.h>
24
25 /* FUNCTION TO SET THE TEXT TO PRINT */
26 #define LCD_print(pos, msg, length, type, time) ({\
27   char text[length + 1];\
28   char padding[LCD_WIDTH + 1];\
29   String setup = type;\
30   \
31   for(int i = 0; i <= LCD_WIDTH; i++) padding[i] = i != LCD_WIDTH ? ' ': '\0';\
32   setup.toLowerCase();\
33   \
34   if (setup == "left") sprintf(text, length + 1, "%s%s", msg, padding);\
35   else {\
36     int space = length - strlen(msg);\
37     if (setup == "center") {\

```

Output Serial Monitor x

Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM4')

Conectando ao WiFi TROJANUBL, aguarde,...

***** WiFi is connected *****

SSID: TROJANUBL

Gateway: 192.168.1.1

IP Address: 192.168.1.16

MAC Address: B0:B2:1C:A7:2A:94

Conexões MQTT rodando no núcleo 0

Tópico: FELMS-MQTT/len_place Mensagem: 5

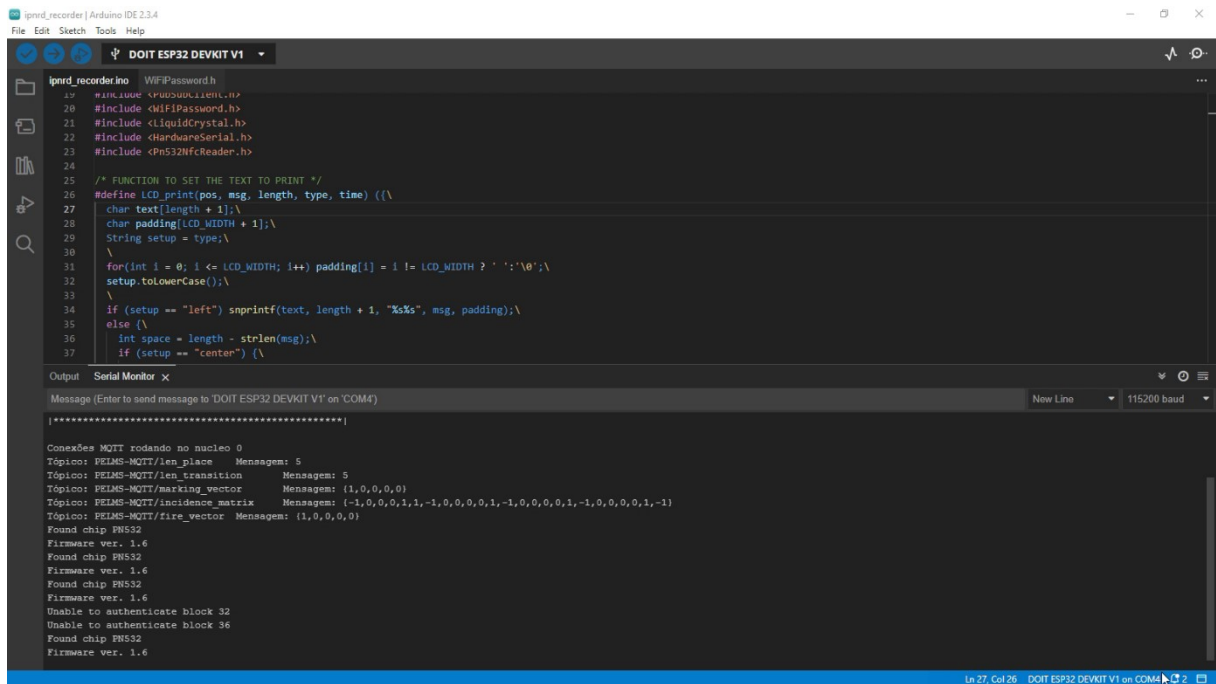
Tópico: FELMS-MQTT/len_transition Mensagem: 5

Tópico: FELMS-MQTT/markig_vector Mensagem: (1,0,0,0,0)

Tópico: FELMS-MQTT/incidence_matrix Mensagem: (-1,0,0,0,1,1,-1,0,0,0,1,-1,0,0,0,1,-1)

Tópico: FELMS-MQTT/fire_vector Mensagem: (1,0,0,0,0)

Figura 61 - Recebimento de dados via MQTT enviadas pelo middleware PELMS-MQTT para o ESP32-Recorder.



```

ipnrd_recorder.ino  WiFiPassword.h
19 #include <PubSubClient.h>
20 #include <WiFiPassword.h>
21 #include <LiquidCrystal.h>
22 #include <HardwareSerial.h>
23 #include <Pn532NfcReader.h>
24
25 /* FUNCTION TO SET THE TEXT TO PRINT */
26 #define LCD_print(pos, msg, length, type, time) ({\
27   char text[length + 1];\
28   char padding[LCD_WIDTH + 1];\
29   String setup = type;\
30   \
31   for(int i = 0; i <= LCD_WIDTH; i++) padding[i] = i != LCD_WIDTH ? ' ': '\0';\
32   setup.toLowerCase();\
33   \
34   if (setup == "left") sprintf(text, length + 1, "%s%s", msg, padding);\
35   else {\
36     int space = length - strlen(msg);\
37     if (setup == "center") {\

```

Output Serial Monitor x

Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM4')

Conexões MQTT rodando no núcleo 0

Tópico: FELMS-MQTT/len_place Mensagem: 5

Tópico: FELMS-MQTT/len_transition Mensagem: 5

Tópico: FELMS-MQTT/markig_vector Mensagem: (1,0,0,0,0)

Tópico: FELMS-MQTT/incidence_matrix Mensagem: (-1,0,0,0,1,1,-1,0,0,0,1,-1,0,0,0,1,-1)

Tópico: FELMS-MQTT/fire_vector Mensagem: (1,0,0,0,0)

Found chip PN532

Firmware ver. 1.6

Found chip PN532

Firmware ver. 1.6

Found chip PN532

Firmware ver. 1.6

Unable to authenticate block 32

Unable to authenticate block 36

Found chip PN532

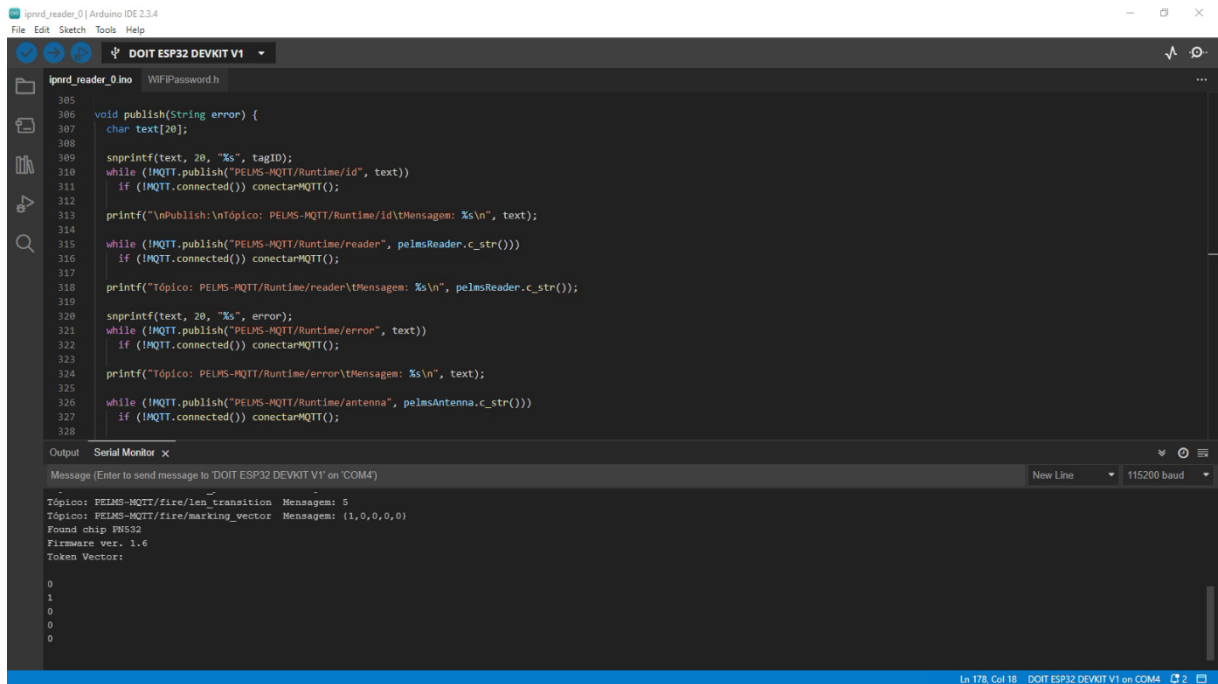
Firmware ver. 1.6

Figura 62 - Salvamento dos dados da iPNRD na Tag no Serial Monitor para o ESP32-Recorder.

```
ipnrd_reader_0.ino  WiFiPassword.h

305
306 void publish(String error) {
307     char text[20];
308
309     snprintf(text, 20, "%s", tagID);
310     while (!MQTT.publish("PELMS-MQTT/Runtime/id", text))
311         if (!MQTT.connected()) connectMQTT();
312
313     printf("\nPublish:\nTópico: PELMS-MQTT/Runtime/id\tMensagem: %s\n", text);
314
315     while (!MQTT.publish("PELMS-MQTT/Runtime/reader", pelmsReader.c_str()))
316         if (!MQTT.connected()) connectMQTT();
317
318     printf("Tópico: PELMS-MQTT/Runtime/reader\tMensagem: %s\n", pelmsReader.c_str());
319
320     snprintf(text, 20, "%s", error);
321     while (!MQTT.publish("PELMS-MQTT/Runtime/error", text))
322         if (!MQTT.connected()) connectMQTT();
323
324     printf("Tópico: PELMS-MQTT/Runtime/error\tMensagem: %s\n", text);
325
326     while (!MQTT.publish("PELMS-MQTT/Runtime/antenna", pelmsAntenna.c_str()))
327         if (!MQTT.connected()) connectMQTT();
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
11
```

Figura 64 - Recebimento de dados via MQTT enviadas pelo middleware PELMS-MQTT para o ESP32-Reader.



The screenshot shows the Arduino IDE interface. The top window displays the code for 'ipnrd_reader_0.ino', which includes MQTT publishing functions for various topics like 'PELMS-MQTT/Runtime/id', 'PELMS-MQTT/Runtime/reader', 'PELMS-MQTT/Runtime/error', and 'PELMS-MQTT/Runtime/antenna'. The bottom window shows the Serial Monitor output, displaying messages such as 'Tópico: PELMS-MQTT/fire/en_transition Mensagem: 5', 'Tópico: PELMS-MQTT/fire/marking_vector Mensagem: (1,0,0,0,0)', 'Found chip PN532', and 'Firmware ver. 1.6'.

```

305
306 void publish(String error) {
307   char text[20];
308
309   snprintf(text, 20, "%s", tagID);
310   while (!MQTT.publish("PELMS-MQTT/Runtime/id", text))
311     if (!MQTT.connected()) conectarMQTT();
312
313   printf("\nPublish: \nTópico: PELMS-MQTT/Runtime/id\tMensagem: %s\n", text);
314
315   while (!MQTT.publish("PELMS-MQTT/Runtime/reader", pelmsReader.c_str()))
316     if (!MQTT.connected()) conectarMQTT();
317
318   printf("Tópico: PELMS-MQTT/Runtime/reader\tMensagem: %s\n", pelmsReader.c_str());
319
320   snprintf(text, 20, "%s", error);
321   while (!MQTT.publish("PELMS-MQTT/Runtime/error", text))
322     if (!MQTT.connected()) conectarMQTT();
323
324   printf("Tópico: PELMS-MQTT/Runtime/error\tMensagem: %s\n", text);
325
326   while (!MQTT.publish("PELMS-MQTT/Runtime/antenna", pelmsAntenna.c_str()))
327     if (!MQTT.connected()) conectarMQTT();
328

```

Output Serial Monitor x

Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM4')

Tópico: PELMS-MQTT/fire/en_transition Mensagem: 5
Tópico: PELMS-MQTT/fire/marking_vector Mensagem: (1,0,0,0,0)
Found chip PN532
Firmware ver. 1.6
Token Vector:
0
1
0
0
0

Ln 178, Col 18 DOIT ESP32 DEVKIT V1 on COM4 2

Figura 65 – Disparo e salvamento dos dados da iPNRD na Tag no Serial Monitor para o ESP32-Reader.

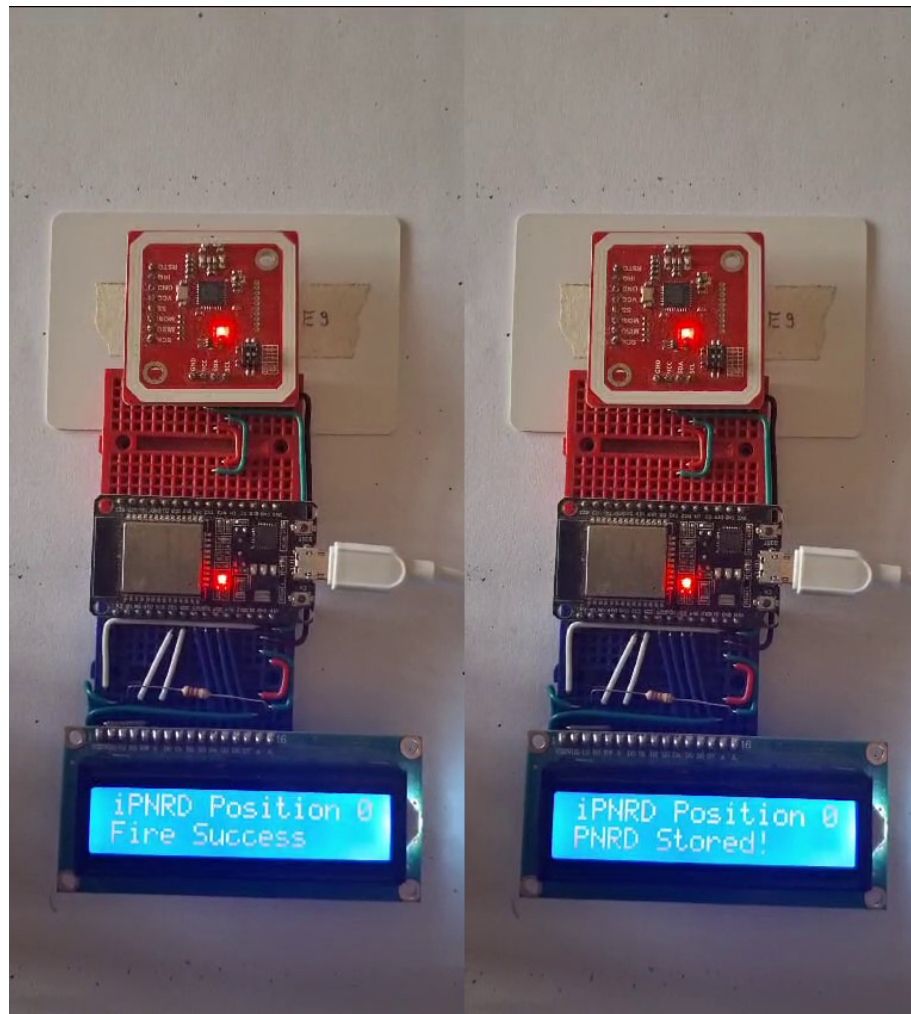


Figura 66 – Disparo e salvamento dos dados da iPNRD na Tag apresentado na tela LCD 16x2.

```

305
306 void publish(String error) {
307   char text[20];
308
309   snprintf(text, 20, "%s", tagID);
310   while (!MQTT.publish("PELMS-MQTT/Runtime/id", text))
311     if (!MQTT.connected()) conectarMQTT();
312
313   printf("\nPublish:\nTópico: PELMS-MQTT/Runtime/id\tMensaje: %s\n", text);
314
315   while (!MQTT.publish("PELMS-MQTT/Runtime/reader", pelmsReader.c_str()))
316     if (!MQTT.connected()) conectarMQTT();
317
318   printf("\nTópico: PELMS-MQTT/Runtime/reader\tMensaje: %s\n", pelmsReader.c_str());
319
320   snprintf(text, 20, "%s", error);
321   while (!MQTT.publish("PELMS-MQTT/Runtime/error", text))
322     if (!MQTT.connected()) conectarMQTT();
323
324   printf("\nTópico: PELMS-MQTT/Runtime/error\tMensaje: %s\n", text);
325
326   while (!MQTT.publish("PELMS-MQTT/Runtime/antenna", pelmsAntenna.c_str()))
327     if (!MQTT.connected()) conectarMQTT();
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Figura 67 - Envio dos dados via MQTT do ESP32-Reader para o middleware PELMS-MQTT referente ao disparo.

PELMS-MQTT

File

Type: iPNRD

Qtd Readers: 5

Reader: Position 0
Qtd Ant:1
IP: 192.168.1.10

Reader: Position 1
Qtd Ant:1
IP: 192.168.1.11

Reader: Position 2
Qtd Ant:1

PNRD: NO_Error

Reader: Position 0

TagId: A405111E

Places: 5

Transitions: 5

Marking Vector

P0	0
P1	1
P2	0
P3	0
P4	0

Incidence Matrix

	P0	P1	P2	P3	P4	
T0	-1	1	0	0	0	transition 0
T1	0	-1	1	0	0	transition 1
T2	0	0	-1	1	0	transition 2
T3	0	0	0	-1	1	transition 3
T4	1	0	0	0	-1	transition 4

TRANSFER PNDR SETUP

GET RUNTIME INFO

TRANSFER INITIAL DATA

SETUP RUNTIME

Figura 68 – Disparo de uma transição no middleware PELMS-MQTT.

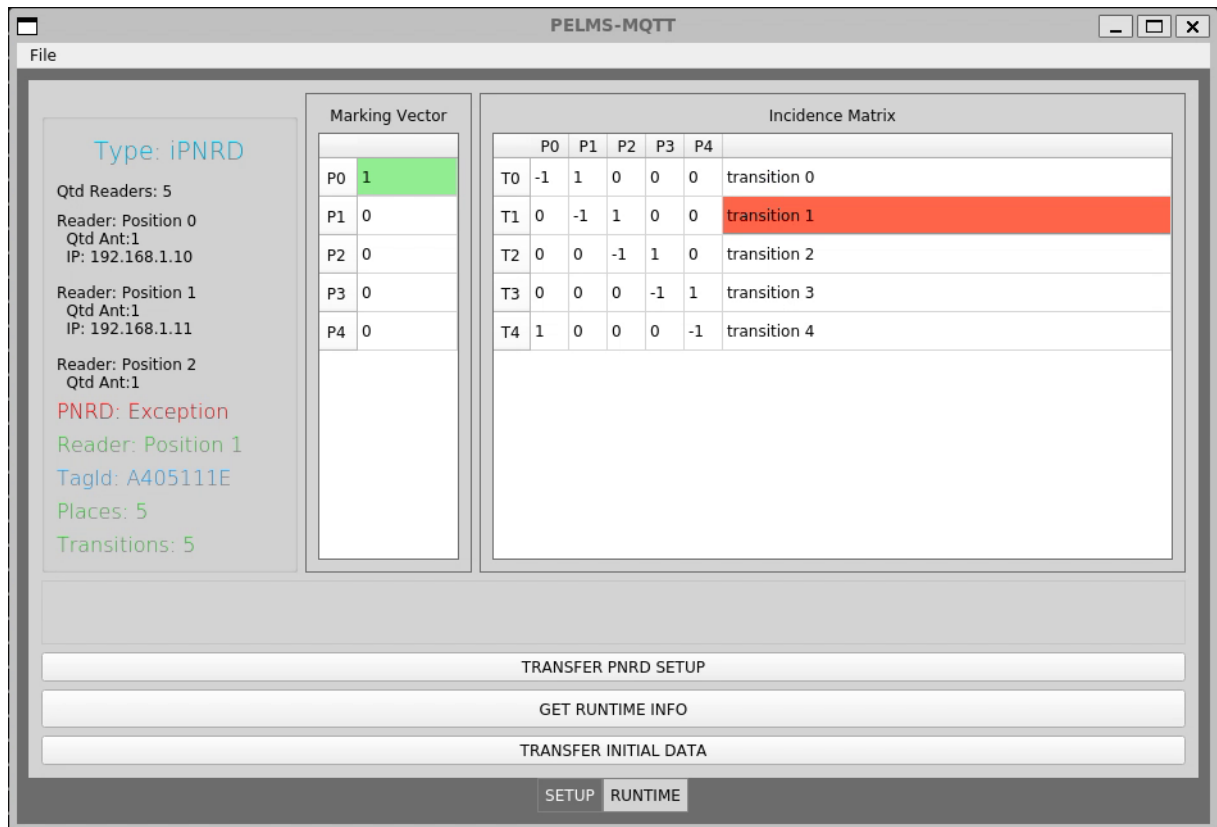


Figura 69 – Excessão de uma Tag no middleware PELMS-MQTT.

5 CONCLUSÃO

Este trabalho apresentou os desafios e as metas necessárias para o desenvolvimento e a implementação do *middleware* PELMS. Para atender os objetivos definidos para o trabalho, demandou muitas horas de estudos e pesquisa, também foi necessário gerar mapas de funcionamento das bibliotecas PNRD e PN532, e do *middleware* PALMS. Realizados as correções e os testes foi possível constatar que o trabalho atendeu as expectativas, porém foi necessário retirar a apresentação e os testes que seriam realizados na placa de circuito impresso desenvolvido no MAPL (ou Laboratório de Planejamento de Automação e Manufatura do inglês *Manufacturing Automated Planning Lab*), o motivo da remoção foi a falta de tempo para confecção do código para utilização da placa.

O ESP32 se mostrou uma plataforma bem robusta e capaz de suportar as necessidades do projeto permitindo a interação com leitores (PN532 NFC Reader V3), LCDs e WiFi, sem a necessidade de módulos extras e cartão SD. A escolha do Broker Mosquitto para o protocolo MQTT atendeu as expectativas de funcionamento e gerou a possibilidade de implementação em rede local sem fio (WLAN - *Local Area Network*) ou em rede de longa distância (WAN – *Wide Area Network*).

A implementação do protocolo MQTT no PELMS e no ESP32 garantiu ao projeto uma característica de um sistema IoT, com comunicação em nuvem e funcionamento em tempo real. Devido a possibilidade do *middleware* PELMS e da plataforma ESP32 comunicar com qualquer Broker, existe uma infinidade de Brokers que podem ser utilizadas, de gratuitos a pagos, e podem ser escalonados de acordo com a necessidade do sistema.

Com o PELMS é possível gerar e gerenciar PNRD/iPNRD, monitorando os disparos e identificando as exceções que podem ocorrer no sistema. Ele possui uma relação 1:N, onde podem ser adicionadas quantos microcontroladores o Broker suportar. No caso de uma falha na conexão, os dados da RP estão salvos no Broker Local, mantendo o sistema ainda ativo.

Os resultados obtidos tanto no *middleware* PELMS quanto na plataforma ESP32 apresentaram dados importantes para a constatação do funcionamento do projeto, permitindo a aplicação em plantas industriais e em Redes de Petri mais complexas. Por ser um trabalho de caráter didático e se tratar de um protótipo, o código do PELMS encontrado no Github do MAPL (<https://github.com/MAPL-UFU/PELMS>) é um software livre e pode ser alterado para usos industriais e acadêmicos.

5.1 Trabalhos Futuros

Para a melhoria e escalabilidade da solução, os trabalhos futuros podem serão apresentados nos itens abaixo.

5.1.1 Hardware

A implementação do PELMS de CLP didático IoT baseado em ESP32 desenvolvida no MAPL será de grande importância visto que ela integra todos os componentes necessários para o funcionamento da PNRD e permite a integração de outros tipos de periféricos.

5.1.2 *Back-end e Front-end*

A implementação de uma versão que utiliza tanto FTP quanto MQTT para a função *runtime* possibilita a confirmação do resultado dos disparos.

A Integração do PELMS-MQTT em uma *Hyperledger* permite a geração de um banco de dados com as informações dos leitores e dos disparos realizados na Rede de Petri.

Devido aos leitores obterem IPs dinâmicos ao conectar à internet, a remoção da entrada do IP dos leitores no *middleware* PELMS é imprescindível, pois não garante a informação correta do IP do leitor.

Para otimizar o funcionamento do Sistema a implementação de um código único para leitura e gravação dos dados da Rede de Petri nas etiquetas, otimiza processo de gravação e disparo delas.

A criação de menus de entrada dos dados de conexão e dos tópicos para a comunicação MQTT, a criação de um aplicativo de celular para a PNRD/iPNRD integrado ao leitor NFC do dispositivo e a implementação de uma animação do funcionamento da planta no *middleware* PELMS será de grande importância, visto que permite que o usuário possa interagir com a Rede de Petri.

REFERÊNCIAS

MURATA, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, IEEE, v. 77, n. 4, p. 541–580, 1989.

TAVARES, J. J.-P. Z. D. S.; SARAIVA, T. A. Elementary petri net inside rfid distributed database (pnrd). *International Journal of Production Research*, Taylor & Francis, v. 48, n. 9, p. 2563–2582, 2010.

FONSECA, J. P. d. S. et al. Redes de petri de alto nível e pnrd invertida associadas ao controle de robôs móveis: uma abordagem para operações de busca e salvamento em trilhas e travessias. Universidade Federal de Uberlândia, 2018.

C.E.A. da Silva, J.J.P.Z.S. Tavares, M.V.M. Ferreira (2018). Arduino Library Developed for Petri Net Inserted into RFID Database and Variants. In: Khomenko, V., Roux, O. (eds) *Application and Theory of Petri Nets and Concurrency. PETRI NETS 2018. Lecture Notes in Computer Science*, vol 10877. Springer, Cham. https://doi.org/10.1007/978-3-319-91268-4_22 Brown, F., Harris, M.G., and Other, A.N. (1998). Name of paper. In Name(s) of editor(s) (ed.), Name of book in italics, page numbers. Publisher, Place of publication.

Arduino® (2024). <www.arduino.cc>. Acessado em 07 de dezembro 2024.

ESP32® (2024). <<https://www.espressif.com/en/products/socs/esp32>>. Acessado em 07 de dezembro de 2024.

PAULA, Roger Henrique Carrijo de. Integração da PNRD com banco de dados distribuído Hyperledger Sawtooth. 2021. 120 f. Trabalho de Conclusão de Curso (Graduação em Engenharia Mecatrônica) – Universidade Federal de Uberlândia, Uberlândia, 2021.

HWANG, A., GONZALEZ, A. G. C., MONTEIRO, J. G. T. Identificação por Rádio Frequência. 2013. Disponível em <https://www.gta.ufrj.br/grad/13_1/rfid/index.html>. Acessado em 07 de dezembro de 2024.

PDA Soluções. Tudo o que você precisa saber sobre a tecnologia RFID. 2021. Disponível em <<https://pdasolucoes.com.br/tecnologia-rfid/>>. Acessado em 07 de dezembro de 2024

Ok, K.; Coskun, V.; Aydin, MN; Ozdenizci, B. Benefícios Atuais e Direções Futuras dos Serviços NFC. Em *Anais da Conferência Internacional IEEE sobre Educação e Tecnologia de Gestão*, Cairo, Egito, 2–4 de novembro de 2010; pp. 334–338.

FUJITSU LTD. Datasheet World's Largest-Capacity 64KByte FRAM Metal Mount RFID

Tag. 2014. Disponível em: <<http://www.fujitsu.com/downloads/AIT/ait-downloads64kbtag.pdf>>. Acesso em 08/12/2017.

MUROFUSHI, R. H., Desenvolvimento de um sistema de posicionamento interno baseado na potência do sinal de resposta de um sistema RFID. 2016. Dissertação de Mestrado, Universidade Federal de Uberlândia, Uberlândia.

AHSAN, K.; SHAH, H.; KINGTON P. RFID Applications: An Introductory and Exploratory Study. IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 1, Nº 3, 2010.

ZELBST, D. P. J.; SOWER, D. V. E. RFID for the Supply Chain and Operations Professional. 3. ed. [S.l.]: Business Expert Press, 2021.

IGOE, T.; COLEMAN, D.; JEPSON, B. Beginning NFC. 1. ed. [S.l.]: O'Reilly Media, Inc, 2014.

SILVA, C. E. A. D. Desenvolvimento de biblioteca para aplicações de pnrp e pnrp invertida embarcadas em arduino. Repositório Universidade Federal de Uberlândia, p. 46–54, 2017.

DAVID, R.; ALLA, H. Discrete, Continuous, and Hybrid Petri Nets. Springer Science & Business Media, 2010.

SILVA, Cassio Moreira. Sistema de controle de apresentação por meio de Raspberry Pi. 2019. 39 f. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação) - Universidade Federal de Uberlândia, Monte Carmelo, 2019. URI: <https://repositorio.ufu.br/handle/123456789/26247>.

SILVA, Gabriel de Oliveira; DIAS, Wanderson Roger Azevedo; ESCUDERO, Danilo Pereira. Análise do Desempenho Computacional da Raspberry Pi Executando o Benchmark SysBench. In: WORKSHOP DE INICIAÇÃO CIENTÍFICA - SIMPÓSIO EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO (SSCAD), 23. , 2022, Florianópolis. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2022 . p. 65-72. DOI: https://doi.org/10.5753/wscad_estendido.2022.226666.

Raspberry Pi. (2024). Raspberry Pi product series explained. Disponível em: <<https://www.raspberrypi.com/news/raspberry-pi-product-series-explained/>>. Acessado em: 07 de dezembro de 2024.

Raspberry Pi. (2024). Raspberry Pi Documentation. Disponível em: <<https://www.raspberrypi.com/documentation/>>. Acessado em: 07 de Dezembro de 2024.

Microsoft. (2016). Windows 10 IoT Core support for Raspberry Pi 3. Disponível em: <<https://blogs.windows.com/windowsdeveloper/2016/02/29/windows-10-iot-core-support-for-raspberry-pi-3/>>. Acessado em: 07 de dezembro de 2024.

SOUZA, R. et al. Internet of Things (IoT) and its Applications in the Industry. In: INTERNATIONAL CONFERENCE ON TECHNOLOGICAL INNOVATION FOR INDUSTRIAL DEVELOPMENT, 3., 2022, São Paulo. Anais... São Paulo: Springer, 2022. p. 012019. DOI: 10.1088/1742-6596/2023/1/012019. Disponível em <<https://iopscience.iop.org/article/10.1088/1742-6596/2023/1/012019/pdf>>. Acessado em: 09 abr. 2024.

SANTOS, P. et al. Emerging Trends in IoT Security: A Review. In: INTERNATIONAL CONFERENCE ON TECHNOLOGICAL INNOVATION FOR INDUSTRIAL DEVELOPMENT, 2., 2020, São Paulo. Anais... São Paulo: Springer, 2020. p. 012044. DOI: 10.1088/1742-6596/2020/1/012044. Disponível em <<https://iopscience.iop.org/article/10.1088/1742-6596/2020/1/012044/pdf>>. Acessado em: 09 abr. 2024.

APÊNDICE A INSTALAÇÃO DO BROKER MOSQUITTO MQTT NA DISTRO LINUX

Os passos para instalação do Broker Mosquitto MQTT na distribuição Linux são:

- Primeiro Passo: Abertura da janela terminal pressionando as teclas OS+T ou Ctrl+Alt+T;
- Segundo Passo: Execução do comando “sudo apt update && sudo apt full-upgrade -y”; (Esse comando atualiza as bibliotecas instaladas no sistema operacional e a lista de bibliotecas disponíveis para instalação)
- Terceiro Passo: Execução do comando “sudo apt install -y mosquitto mosquitto-clients”; (Esse comando instala o servidor/Broker Mosquito MQTT e o cliente mosquito)

APÊNDICE B SETUP DO BROKER MOSQUITTO MQTT NA DISTRO LINUX

Serão apresentados alguns setups que foram implementados no Raspberry Pi para o projeto e no servidor Debian disponível no Laboratório de Planejamento de Automação e Manufatura (MAPL – *Manufacturing Automated Planning Lab*) para funcionamento do Broker. Os passos para setup inicial do Broker Mosquitto MQTT são:

- Primeiro Passo: Abra a janela terminal pressionando as teclas OS+T ou Ctrl+Alt+T;
- Segundo Passo: Execute o comando “*sudo nano /etc/mosquitto/mosquitto.conf*”; (Esse comando abre o arquivo de configuração do Broker no formato *superuser*)
- Terceiro Passo: No final do arquivo insira os textos “*listener 1883*” e “*allow_anonymous true*”; (Essas linhas configuram a porta de acesso e permite acesso de usuários anônimos, ou seja, usuários que não querem se identificar)

```
listener 1883

# To setup a user access in broker you have to change the allow_anonymous to false and set the password_file
allow_anonymous true
#password_file <pw-setup-file>
```

Figura 70 - Setup da porta e tipo de acesso no Broker.

- Quarto Passo: Caso não deseje realizar o bridge entre dois Brokers pule para o último passo. No Broker local, ou seja, no Broker que não terá acesso externamente insira os textos “*connection <name of the bridge connection>*”, “*address <address>:<port>*”, “*remote_username < username>*”, “*remote_password <password>*” e “*topic <topic> <type> <QoS>*”.
 - *<name of the bridge connection>*: neste parâmetro será nomeado o bridge;
 - *<address>* e *<port>*: nestes parâmetros são inseridos o endereço e a porta de conexão, caso a porta seja 1883 não é necessário inserir o parâmetro *<port>*;
 - *<username>* e *<password>*: nestes parâmetros são inseridos o usuário e senha para acesso ao broker externo, caso o broker externo aceite acesso anônimo não é necessário inserir os textos “*remote_username < username>*” e “*remote_password <password>*”;
 - *<topic>*: neste parâmetro é inserido o nome do tópico que deseja publicar ou subscrever, caso seja necessário publicar e subscrever todos os tópicos insira o texto “#”;
 - *<type>*: neste parâmetro é inserido o tipo de bridge que será formado, subscrito e publicador “both”, somente subscritor “in” ou somente publicador “out”;

- *<QoS>*: neste parâmetro é inserido a qualidade de serviço do bridge “0”, “1” ou “2”, conforme foi explicado no item 2.11.

```
# Brige configuration to publish and subscribe information in the broker
connection Brige-do
address 200.19.144.16:1883

# Setup remote username and password of the broker (if exist)
#remote_username <broker-username>
#remote_password <broker-password>

topic # both 0
#topic # out 0
#topic # in 0
```

Figura 71 - Setup do brige entre dois Broker.

- Quinto Passo: Salve e feche o arquivo de setup do Broker pressionando as teclas “Ctrl+X”, em seguida “Y” e depois “Enter”;

APÊNDICE C MAINWINDOW.PY

```
#!/usr/bin/python3
"""
Created on Nov, 2021
Updated on:
    * Jan, 2022
    * Dec, 2024
@author: Daniel Barbosa Pereira
@updater author:
    * Roger Henrique Carrijo de Paula
    * Daniel Barbosa Pereira
    * Álisson Carvalho Vasconcelos
@Github authors:
    * rogercarrijo
    * daniel-b-pereira
    * AlissonCV
"""

# Importing functions inside PyQt5
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

#-----
from os import getcwd as path
import sys

# Function to get de directory
def resource_path(relative_path):
    if 'gui' in path():
        base_path = path() + relative_path
    else:
        base_path = path() + "/gui" + relative_path

    if (sys.platform == 'win32'):
        base_path = base_path.replace("/", "\\")

    return base_path
#-----

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(931, 609)
        MainWindow.move(100, 100)

        sizePolicy = QSizePolicy(QSizePolicy.Fixed, QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
```

```

sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(MainWindow.sizePolicy().hasHeightForWidth())

MainWindow.setSizePolicy(sizePolicy)
MainWindow.setMinimumSize(QSize(50, 50))
MainWindow.setStyleSheet(
    #Window Layout Color
    "QMainWindow {\n"
    "    background-color: DimGray;\n"
    "}\n"

    #Panel Background Color
    "QTabWidget {\n"
#    "    background-color: rgb(255, 255, 255);\n"
    "    background-color: transparent;\n"
#    "    border: 1px solid rgb(221, 216, 216);\n"
    "}\n"

    #Panel Layout Design
    "QTabWidget::pane {\n"
    "    background-color: LightGray;\n"
    "    border: 1px solid Dark;\n"
    "}\n"

    #Tab Bar Background Color
    "QTabWidget::tab-bar {\n"
    "    background-color: rgb(238, 238, 236);\n"
    "    border: 1px solid rgb(221, 216, 216);\n"
    "    alignment: center;\n"
    "}\n"

    #Tab Bar Layout Design
    "QTabBar::tab {\n"
    "    color: White;\n"
    "    background-color: DimGray;\n"
    "    border: 1px solid Gray;\n"
    "    padding: 5px;\n"
    "    margin-bottom: 0px;\n"
    "}\n"

    #Tab Bar Selected Layout Color
    "QTabBar::tab:selected {\n"
    "    background-color: LightGray;\n"
    "    color: Dark;\n"
    "}\n"

    #Markin Vector and Incidence Matrix Box Layout Color
    "QTableWidget {\n"
    "    background-color: White;\n"
    "    border: 1px solid DimGray;\n"

```

```

"}\n"

#
"QWidget::section {\n"
"  background-color: transparent;\n"
"  border:0px solid transparent;\n"
"}\n"

#Header of Markin Vector and Incidence Matrix
"QHeaderView {\n"
"  background-color: white;\n"
"}\n"

#
"QHeaderView::section {\n"
"  background-color: white;\n"
"}\n"

#
"QHeaderView::section:vertical {\n"
"  background-color: white;\n"
"}\n"

#
"QHeaderView::section:horizontal {\n"
"  background-color:rgb(238, 238, 236);\n"
"}\n"

#Marking Vector and Incidence Matrix Menu Layout Color
"QGroupBox#groupBoxMatrix,
QGroupBox#markingGroupBox,QGroupBox#groupBoxMatrix2,
QGroupBox#markingGroupBox2 {\n"
"  border: 1px solid DimGray;\n"
"  background-color: transparent;\n"
"}\n"

#
# "QGroupBox#groupBox_Labels{\n"
# "  background-color: white;\n"
# "  border:0px ,solid ,rgb(186, 189, 182);\n"
# "}\n"

#Setup Type Layout Color
"#setupBg_groupBox {\n"
"  background-color: LightGray;\n"
"  border:1px solid DimGray;\n"
"}\n"

#
"#pelmsType_label {\n"

```

```

" color:rgb(2, 160, 200);\n"
" background-color: transparent;\n"
" font: 75 14pt \"Courier 10 Pitch\";\n"
"}\n"

#
"#exception_label {\n"
" color:rgb(204, 0, 0);\n"
" background-color: transparent;\n"
" font: 75 12pt \"Courier 10 Pitch\";\n"
"}\n"

#
"#reader_label {\n"
" color:rgb(60, 167, 61);\n"
" background-color: transparent;\n"
" font: 75 12pt \"Courier 10 Pitch\";\n"
"}\n"

#
"#id_label {\n"
" color:rgb(48, 140, 198);\n"
" background-color: transparent;\n"
" font: 75 12pt \"Courier 10 Pitch\";\n"
"}\n"

#
"#places_label {\n"
" color:rgb(60, 167, 61);\n"
" background-color: transparent;\n"
" font: 75 12pt \"Courier 10 Pitch\";\n"
"}\n"

#
"#transitions_label {\n"
" color:rgb(60, 167, 61);\n"
" background-color: transparent;\n"
" font: 75 12pt \"Courier 10 Pitch\";\n"
"}\n"

#
"#info_label {\n"
" color:rgb(204, 0, 0);\n"
" background-color: transparent;\n"
" font: 75 12pt \"Courier 10 Pitch\";\n"
"}\n"

#Text Box of Create Setup File Layout Design
"#setupInfo_label {\n"
" background-color: white;\n"

```

```

" border: 1px solid DimGray;\n"
" color:rgb(60, 167, 61);\n"
" font: 75 12pt \"Courier 10 Pitch\";\n"
"}\n"

#CREATE SETUP FILE
"#createSetup_pushButton {\n"
" background-color: DimGray;\n"
" color: White;\n"
" border: 1px solid Dark;\n"
" font: 75 12pt \"Courier 10 Pitch\";\n"
" font-weight:bold;\n"
"}\n"

#Add
"#addIP_pushButton {\n"
" background-color: DimGray;\n"
" color: White;\n"
" border: 1px solid Dark;\n"
" font: 75 12pt \"Courier 10 Pitch\";\n"
" font-weight:bold;\n"
"}\n"
)
self.centralwidget = QWidget(MainWindow)
self.centralwidget.setObjectName("centralwidget")
self.verticalLayout_11 = QVBoxLayout(self.centralwidget)
self.verticalLayout_11.setObjectName("verticalLayout_11")
self.setup_tabWidget = QTabWidget(self.centralwidget)
self.setup_tabWidget.setTabPosition(QTabWidget.South)
self.setup_tabWidget.setTabShape(QTabWidget.Rounded)
self.setup_tabWidget.setObjectName("setup_tabWidget")
self.tab_setup = QWidget()
self.tab_setup.setObjectName("tab_setup")
self.verticalLayout_5 = QVBoxLayout(self.tab_setup)
self.verticalLayout_5.setObjectName("verticalLayout_5")
self.horizontalLayout_2 = QHBoxLayout()
self.horizontalLayout_2.setObjectName("horizontalLayout_2")
self.verticalLayout_4 = QVBoxLayout()
self.verticalLayout_4.setObjectName("verticalLayout_4")
self.label_5 = QLabel(self.tab_setup)
self.label_5.setObjectName("label_5")
self.label_5.setMaximumSize(QSize(230, 120))
# self.label_5.setFrameShape(QFrame.NoFrame)
self.label_5.setText("")
self.label_5.setPixmap(QPixmap(resource_path("images/femec - Copia.jpeg")))
self.label_5.setScaledContents(True)
self.label_5.setAlignment(Qt.AlignLeft)
# self.label_5.setWordWrap(False)
self.verticalLayout_4.addWidget(self.label_5)
self.horizontalLayout_6 = QHBoxLayout()

```

```

self.horizontalLayout_6.setContentsMargins(0, 0, -1, -1)
self.horizontalLayout_6.setSpacing(6)
self.horizontalLayout_6.setObjectName("horizontalLayout_6")
self.bgPnrd_verticalLayout = QVBoxLayout()
self.bgPnrd_verticalLayout.setContentsMargins(0, 0, 0, 0)
self.bgPnrd_verticalLayout.setObjectName("bgPnrd_verticalLayout")
self.setupBg_groupBox = QGroupBox(self.tab_setup)
self.setupBg_groupBox.setMaximumSize(QSize(100, 16777215))
self.setupBg_groupBox.setTitle("")
self.setupBg_groupBox.setAlignment(Qt.AlignCenter)
self.setupBg_groupBox.setObjectName("setupBg_groupBox")
self.verticalLayout_8 = QVBoxLayout(self.setupBg_groupBox)
self.verticalLayout_8.setObjectName("verticalLayout_8")
self.label_6 = QLabel(self.setupBg_groupBox)
self.label_6.setAlignment(Qt.AlignCenter)
self.label_6.setObjectName("label_6")
self.verticalLayout_8.addWidget(self.label_6)
self.setupPelms_comboBox = QComboBox(self.setupBg_groupBox)
self.setupPelms_comboBox.setMinimumSize(QSize(0, 60))
self.setupPelms_comboBox.setMaximumSize(QSize(100, 16777215))
font = QFont()
font.setKerning(True)
self.setupPelms_comboBox.setFont(font)
self.setupPelms_comboBox.setLayoutDirection(Qt.LeftToRight)
self.setupPelms_comboBox.setObjectName("setupPelms_comboBox")
self.setupPelms_comboBox.addItem("")
self.setupPelms_comboBox.addItem("")
self.verticalLayout_8.addWidget(self.setupPelms_comboBox)
self.bgPnrd_verticalLayout.addWidget(self.setupBg_groupBox)
self.horizontalLayout_6.addLayout(self.bgPnrd_verticalLayout)
self.verticalLayout_9 = QVBoxLayout()
self.verticalLayout_9.setContentsMargins(-1, 0, -1, -1)
self.verticalLayout_9.setObjectName("verticalLayout_9")
self.horizontalLayout_9 = QHBoxLayout()
self.horizontalLayout_9.setContentsMargins(-1, 0, -1, -1)
self.horizontalLayout_9.setObjectName("horizontalLayout_9")
self.label_7 = QLabel(self.tab_setup)
self.label_7.setMaximumSize(QSize(100, 16777215))
self.label_7.setObjectName("label_7")
self.horizontalLayout_9.addWidget(self.label_7)
self.readerName_lineEdit = QLineEdit(self.tab_setup)
self.readerName_lineEdit.setMaximumSize(QSize(300, 16777215))
self.readerName_lineEdit.setObjectName("readerName_lineEdit")
self.horizontalLayout_9.addWidget(self.readerName_lineEdit)
self.verticalLayout_9.addLayout(self.horizontalLayout_9)
self.horizontalLayout_10 = QHBoxLayout()
self.horizontalLayout_10.setObjectName("horizontalLayout_10")
self.label_8 = QLabel(self.tab_setup)
self.label_8.setObjectName("label_8")
self.horizontalLayout_10.addWidget(self.label_8)

```

```

self.qtdTotalTansitions_label = QLabel(self.tab_setup)
self.qtdTotalTansitions_label.setText("")
self.qtdTotalTansitions_label.setObjectName("qtdTotalTansitions_label")
self.horizontalLayout_10.addWidget(self.qtdTotalTansitions_label)
self.nAntennas_spinBox = QSpinBox(self.tab_setup)
self.nAntennas_spinBox.setObjectName("nAntennas_spinBox")
self.horizontalLayout_10.addWidget(self.nAntennas_spinBox)
self.verticalLayout_9.addLayout(self.horizontalLayout_10)
self.label_3 = QLabel(self.tab_setup)
self.label_3.setMaximumSize(QSize(400, 30))
self.label_3.setAlignment(Qt.AlignCenter)
self.label_3.setObjectName("label_3")
self.verticalLayout_9.addWidget(self.label_3)
self.IP_lineEdit = QLineEdit(self.tab_setup)
self.IP_lineEdit.setObjectName("IP_lineEdit")
self.verticalLayout_9.addWidget(self.IP_lineEdit)
self.addIP_pushButton = QPushButton(self.tab_setup)
self.addIP_pushButton.setMinimumSize(QSize(0, 40))
self.addIP_pushButton.setMaximumSize(QSize(400, 16777215))
self.addIP_pushButton.setObjectName("addIP_pushButton")
self.verticalLayout_9.addWidget(self.addIP_pushButton)
self.horizontalLayout_6.addLayout(self.verticalLayout_9)
self.verticalLayout_4.addLayout(self.horizontalLayout_6)
self.setupInfo_label = QLabel(self.tab_setup)
self.setupInfo_label.setMaximumSize(QSize(500, 16777215))
self.setupInfo_label.setObjectName("setupInfo_label")
self.verticalLayout_4.addWidget(self.setupInfo_label)
self.createSetup_pushButton = QPushButton(self.tab_setup)
self.createSetup_pushButton.setMinimumSize(QSize(0, 50))
self.createSetup_pushButton.setMaximumSize(QSize(500, 16777215))
self.createSetup_pushButton.setObjectName("createSetup_pushButton")
self.verticalLayout_4.addWidget(self.createSetup_pushButton)
self.horizontalLayout_2.addLayout(self.verticalLayout_4)
self.markingGroupBox = QGroupBox(self.tab_setup)
self.markingGroupBox.setMinimumSize(QSize(100, 0))
self.markingGroupBox.setMaximumSize(QSize(130, 16777215))
self.markingGroupBox.setLayoutDirection(Qt.LeftToRight)
self.markingGroupBox.setTitle("")
self.markingGroupBox.setAlignment(Qt.AlignCenter)
self.markingGroupBox.setObjectName("markingGroupBox")
self.verticalLayout_2 = QVBoxLayout(self.markingGroupBox)
self.verticalLayout_2.setObjectName("verticalLayout_2")
self.marking_vector = QLabel(self.markingGroupBox)
self.marking_vector.setMinimumSize(QSize(30, 30))
self.marking_vector.setMaximumSize(QSize(120, 16777215))
self.marking_vector.setLayoutDirection(Qt.LeftToRight)
self.marking_vector.setAlignment(Qt.AlignCenter)
self.marking_vector.setObjectName("marking_vector")
self.verticalLayout_2.addWidget(self.marking_vector)
self.markingVector_tw = QTableWidgetItem(self.markingGroupBox)

```

```

self.markingVector_tw.setMinimumSize(QSize(30, 0))
self.markingVector_tw.setMaximumSize(QSize(120, 16777215))
self.markingVector_tw.setRowCount(0)
self.markingVector_tw.setColumnCount(0)
self.markingVector_tw.setObjectName("markingVector_tw")
self.markingVector_tw.horizontalHeader().setDefaultSectionSize(30)
self.markingVector_tw.horizontalHeader().setMinimumSectionSize(30)
self.markingVector_tw.horizontalHeader().setStretchLastSection(True)
self.markingVector_tw.verticalHeader().setMinimumSectionSize(30)
self.verticalLayout_2.addWidget(self.markingVector_tw)
self.horizontalLayout_2.addWidget(self.markingGroupBox)
self.groupBoxMatrix = QGroupBox(self.tab_setup)
self.groupBoxMatrix.setMinimumSize(QSize(350, 0))
self.groupBoxMatrix.setTitle("")
self.groupBoxMatrix.setObjectName("groupBoxMatrix")
self.verticalLayout = QVBoxLayout(self.groupBoxMatrix)
self.verticalLayout.setObjectName("verticalLayout")
self.label = QLabel(self.groupBoxMatrix)
self.label.setMaximumSize(QSize(16777215, 30))
self.label.setAlignment(Qt.AlignCenter)
self.label.setObjectName("label")
self.verticalLayout.addWidget(self.label)
self.incMatrix_tw = QTableWidgetItem(self.groupBoxMatrix)
sizePolicy = QSizePolicy(QSizePolicy.Preferred, QSizePolicy.Preferred)
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.incMatrix_tw.sizePolicy().hasHeightForWidth())
self.incMatrix_tw.setSizePolicy(sizePolicy)
self.incMatrix_tw.setMinimumSize(QSize(250, 0))
self.incMatrix_tw.setMaximumSize(QSize(16777215, 16777215))

self.incMatrix_tw.setSizeAdjustPolicy(QAbstractScrollArea.AdjustToContentsOnFirstShow)

self.incMatrix_tw.setAutoScroll(True)
self.incMatrix_tw.setGridStyle(Qt.SolidLine)
self.incMatrix_tw.setRowCount(0)
self.incMatrix_tw.setColumnCount(0)
self.incMatrix_tw.setObjectName("incMatrix_tw")
self.incMatrix_tw.horizontalHeader().setVisible(False)
self.incMatrix_tw.horizontalHeader().setCascadingSectionResizes(False)
self.incMatrix_tw.horizontalHeader().setDefaultSectionSize(30)
self.incMatrix_tw.horizontalHeader().setMinimumSectionSize(30)
self.incMatrix_tw.horizontalHeader().setSortIndicatorShown(False)
self.incMatrix_tw.horizontalHeader().setStretchLastSection(True)
self.incMatrix_tw.verticalHeader().setDefaultSectionSize(30)
self.incMatrix_tw.verticalHeader().setMinimumSectionSize(30)
self.incMatrix_tw.verticalHeader().setStretchLastSection(False)
self.verticalLayout.addWidget(self.incMatrix_tw)
self.horizontalLayout_2.addWidget(self.groupBoxMatrix)
self.verticalLayout_5.addLayout(self.horizontalLayout_2)

```

```

self.horizontalLayout_5 = QHBoxLayout()
self.horizontalLayout_5.setContentsMargins(-1, 0, -1, -1)
self.horizontalLayout_5.setObjectName("horizontalLayout_5")
self.verticalLayout_6 = QVBoxLayout()
self.verticalLayout_6.setContentsMargins(-1, 0, -1, -1)
self.verticalLayout_6.setObjectName("verticalLayout_6")
self.label_4 = QLabel(self.tab_setup)
self.label_4.setLayoutDirection(Qt.LeftToRight)
self.label_4.setAlignment(Qt.AlignCenter)
self.label_4.setObjectName("label_4")
self.verticalLayout_6.addWidget(self.label_4)
self.matrix_array = QLineEdit(self.tab_setup)
self.matrix_array.setMinimumSize(QSize(500, 40))
self.matrix_array.setAlignment(Qt.AlignCenter)
self.matrix_array.setObjectName("matrix_array")
self.verticalLayout_6.addWidget(self.matrix_array)
self.horizontalLayout_5.addLayout(self.verticalLayout_6)
self.verticalLayout_7 = QVBoxLayout()
self.verticalLayout_7.setContentsMargins(0, 0, -1, -1)
self.verticalLayout_7.setObjectName("verticalLayout_7")
self.label_2 = QLabel(self.tab_setup)
self.label_2.setAlignment(Qt.AlignCenter)
self.label_2.setObjectName("label_2")
self.verticalLayout_7.addWidget(self.label_2)
self.marking_array = QLineEdit(self.tab_setup)
self.marking_array.setMinimumSize(QSize(0, 40))
self.marking_array.setText("")
self.marking_array.setAlignment(Qt.AlignCenter)
self.marking_array.setObjectName("marking_array")
self.verticalLayout_7.addWidget(self.marking_array)
self.horizontalLayout_5.addLayout(self.verticalLayout_7)
self.verticalLayout_5.addLayout(self.horizontalLayout_5)
self.setup_tabWidget.addTab(self.tab_setup, "")
self.tab_petrinet = QWidget()
self.tab_petrinet.setObjectName("tab_petrinet")
self.verticalLayout_10 = QVBoxLayout(self.tab_petrinet)
self.verticalLayout_10.setObjectName("verticalLayout_10")
self.horizontalLayout = QHBoxLayout()
self.horizontalLayout.setObjectName("horizontalLayout")
self.verticalLayout_15 = QVBoxLayout()
self.verticalLayout_15.setObjectName("verticalLayout_15")
self.groupBox_Labels = QGroupBox(self.tab_petrinet)
self.groupBox_Labels.setMinimumSize(QSize(200, 0))
self.groupBox_Labels.setMaximumSize(QSize(9999999, 16777215))
self.groupBox_Labels.setTitle("")
self.groupBox_Labels.setObjectName("groupBox_Labels")
self.verticalLayout_3 = QVBoxLayout(self.groupBox_Labels)
self.verticalLayout_3.setObjectName("verticalLayout_3")
self.pelmsType_label = QLabel(self.groupBox_Labels)
self.pelmsType_label.setMinimumSize(QSize(0, 30))

```

```

self.pelmsType_label.setMaximumSize(QSize(16777215, 16777215))
font = QFont()
font.setFamily("Courier 10 Pitch")
font.setPointSize(14)
font.setBold(False)
font.setItalic(False)
font.setWeight(9)
self.pelmsType_label.setFont(font)
self.pelmsType_label.setText("")
self.pelmsType_label.setAlignment(Qt.AlignCenter)
self.pelmsType_label.setObjectName("pelmsType_label")
self.verticalLayout_3.addWidget(self.pelmsType_label)
self.qtdReader_label = QLabel(self.groupBox_Labels)
self.qtdReader_label.setMinimumSize(QSize(0, 20))
self.qtdReader_label.setText("")
self.qtdReader_label.setObjectName("qtdReader_label")
self.verticalLayout_3.addWidget(self.qtdReader_label)
self.readerList_label = QLabel(self.groupBox_Labels)
self.readerList_label.setObjectName("readerList_label")
self.verticalLayout_3.addWidget(self.readerList_label)
self.exception_label = QLabel(self.groupBox_Labels)
self.exception_label.setMinimumSize(QSize(0, 20))
self.exception_label.setText("")
self.exception_label.setObjectName("exception_label")
self.verticalLayout_3.addWidget(self.exception_label)
self.reader_label = QLabel(self.groupBox_Labels)
self.reader_label.setMinimumSize(QSize(0, 20))
self.reader_label.setText("")
self.reader_label.setObjectName("reader_label")
self.verticalLayout_3.addWidget(self.reader_label)
self.id_label = QLabel(self.groupBox_Labels)
self.id_label.setMinimumSize(QSize(0, 20))
self.id_label.setText("")
self.id_label.setObjectName("id_label")
self.verticalLayout_3.addWidget(self.id_label)
self.places_label = QLabel(self.groupBox_Labels)
self.places_label.setMinimumSize(QSize(0, 20))
self.places_label.setMaximumSize(QSize(16777215, 16777215))
self.places_label.setAlignment(Qt.AlignLeading|Qt.AlignLeft|Qt.AlignVCenter)
self.places_label.setObjectName("places_label")
self.verticalLayout_3.addWidget(self.places_label)
self.transitions_label = QLabel(self.groupBox_Labels)
self.transitions_label.setMinimumSize(QSize(0, 20))
self.transitions_label.setAlignment(Qt.AlignLeading|Qt.AlignLeft|Qt.AlignVCenter)
self.transitions_label.setObjectName("transitions_label")
self.verticalLayout_3.addWidget(self.transitions_label)
self.verticalLayout_15.addWidget(self.groupBox_Labels)
self.horizontalLayout.addLayout(self.verticalLayout_15)
self.markingGroupBox2 = QGroupBox(self.tab_petrinet)
self.markingGroupBox2.setMinimumSize(QSize(100, 0))

```

```

self.markingGroupBox2.setMaximumSize(QSize(130, 16777215))
self.markingGroupBox2.setTitle("")
self.markingGroupBox2.setObjectName("markingGroupBox2")
self.verticalLayout_13 = QVBoxLayout(self.markingGroupBox2)
self.verticalLayout_13.setObjectName("verticalLayout_13")
self.label_9 = QLabel(self.markingGroupBox2)
self.label_9.setAlignment(Qt.AlignCenter)
self.label_9.setObjectName("label_9")
self.verticalLayout_13.addWidget(self.label_9)
self.markingVector2_tw = QTableWidgetItem(self.markingGroupBox2)
self.markingVector2_tw.setMaximumSize(QSize(120, 16777215))
self.markingVector2_tw.setObjectName("markingVector2_tw")
self.markingVector2_tw.setColumnCount(0)
self.markingVector2_tw.setRowCount(0)
self.markingVector2_tw.horizontalHeader().setDefaultSectionSize(30)
self.markingVector2_tw.horizontalHeader().setMinimumSectionSize(30)
self.markingVector2_tw.horizontalHeader().setStretchLastSection(True)
self.markingVector2_tw.verticalHeader().setMinimumSectionSize(30)
self.verticalLayout_13.addWidget(self.markingVector2_tw)
self.horizontalLayout.addWidget(self.markingGroupBox2)
self.groupBoxMatrix2 = QGroupBox(self.tab_petrinet)
self.groupBoxMatrix2.setMinimumSize(QSize(350, 0))
self.groupBoxMatrix2.setTitle("")
self.groupBoxMatrix2.setObjectName("groupBoxMatrix2")
self.verticalLayout_14 = QVBoxLayout(self.groupBoxMatrix2)
self.verticalLayout_14.setObjectName("verticalLayout_14")
self.label_10 = QLabel(self.groupBoxMatrix2)
self.label_10.setAlignment(Qt.AlignCenter)
self.label_10.setObjectName("label_10")
self.verticalLayout_14.addWidget(self.label_10)
self.incMatrix2_tw = QTableWidgetItem(self.groupBoxMatrix2)
self.incMatrix2_tw.setMinimumSize(QSize(250, 0))
self.incMatrix2_tw.setObjectName("incMatrix2_tw")
self.incMatrix2_tw.setColumnCount(0)
self.incMatrix2_tw.setRowCount(0)
self.incMatrix2_tw.horizontalHeader().setDefaultSectionSize(30)
self.incMatrix2_tw.horizontalHeader().setMinimumSectionSize(30)
self.incMatrix2_tw.horizontalHeader().setStretchLastSection(True)
self.incMatrix2_tw.verticalHeader().setMinimumSectionSize(30)
self.verticalLayout_14.addWidget(self.incMatrix2_tw)
self.horizontalLayout.addWidget(self.groupBoxMatrix2)
self.verticalLayout_10.addLayout(self.horizontalLayout)
self.runtimeTerminal_label = QLabel(self.tab_petrinet)
self.runtimeTerminal_label.setEnabled(True)
self.runtimeTerminal_label.setMinimumSize(QSize(0, 50))
self.runtimeTerminal_label.setMaximumSize(QSize(16777215, 55))
palette = QPalette()
brush = QBrush(QColor(60, 167, 61))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Active, QPalette.WindowText, brush)

```

```

brush = QBrush(QColor(255, 255, 255))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Active, QPalette.Button, brush)
brush = QBrush(QColor(255, 255, 255))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Active, QPalette.Light, brush)
brush = QBrush(QColor(255, 255, 255))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Active, QPalette.Midlight, brush)
brush = QBrush(QColor(127, 127, 127))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Active, QPalette.Dark, brush)
brush = QBrush(QColor(170, 170, 170))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Active, QPalette.Mid, brush)
brush = QBrush(QColor(60, 167, 61))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Active, QPalette.Text, brush)
brush = QBrush(QColor(255, 255, 255))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Active, QPalette.BrightText, brush)
brush = QBrush(QColor(60, 167, 61))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Active, QPalette.ButtonText, brush)
brush = QBrush(QColor(255, 255, 255))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Active, QPalette.Base, brush)
brush = QBrush(QColor(255, 255, 255))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Active, QPalette.Window, brush)
brush = QBrush(QColor(0, 0, 0))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Active, QPalette.Shadow, brush)
brush = QBrush(QColor(255, 255, 255))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Active, QPalette.AlternateBase, brush)
brush = QBrush(QColor(255, 255, 255))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Active, QPalette.ToolTipBase, brush)
brush = QBrush(QColor(0, 0, 0))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Active, QPalette.ToolTipText, brush)
brush = QBrush(QColor(60, 167, 61))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Inactive, QPalette.WindowText, brush)
brush = QBrush(QColor(255, 255, 255))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Inactive, QPalette.Button, brush)
brush = QBrush(QColor(255, 255, 255))
brush.setStyle(Qt.SolidPattern)

```

```

palette.setBrush(QPalette.Inactive, QPalette.Light, brush)
brush = QBrush(QColor(255, 255, 255))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Inactive, QPalette.Midlight, brush)
brush = QBrush(QColor(127, 127, 127))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Inactive, QPalette.Dark, brush)
brush = QBrush(QColor(170, 170, 170))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Inactive, QPalette.Mid, brush)
brush = QBrush(QColor(60, 167, 61))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Inactive, QPalette.Text, brush)
brush = QBrush(QColor(255, 255, 255))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Inactive, QPalette.BrightText, brush)
brush = QBrush(QColor(60, 167, 61))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Inactive, QPalette.ButtonText, brush)
brush = QBrush(QColor(255, 255, 255))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Inactive, QPalette.Base, brush)
brush = QBrush(QColor(255, 255, 255))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Inactive, QPalette.Window, brush)
brush = QBrush(QColor(0, 0, 0))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Inactive, QPalette.Shadow, brush)
brush = QBrush(QColor(255, 255, 255))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Inactive, QPalette.AlternateBase, brush)
brush = QBrush(QColor(255, 255, 255))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Inactive, QPalette.ToolTipBase, brush)
brush = QBrush(QColor(0, 0, 0))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Inactive, QPalette.ToolTipText, brush)
brush = QBrush(QColor(127, 127, 127))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Disabled, QPalette.WindowText, brush)
brush = QBrush(QColor(255, 255, 255))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Disabled, QPalette.Button, brush)
brush = QBrush(QColor(255, 255, 255))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Disabled, QPalette.Light, brush)
brush = QBrush(QColor(255, 255, 255))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Disabled, QPalette.Midlight, brush)
brush = QBrush(QColor(127, 127, 127))

```

```

brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Disabled, QPalette.Dark, brush)
brush = QBrush(QColor(170, 170, 170))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Disabled, QPalette.Mid, brush)
brush = QBrush(QColor(127, 127, 127))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Disabled, QPalette.Text, brush)
brush = QBrush(QColor(255, 255, 255))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Disabled, QPalette.BrightText, brush)
brush = QBrush(QColor(127, 127, 127))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Disabled, QPalette.ButtonText, brush)
brush = QBrush(QColor(255, 255, 255))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Disabled, QPalette.Base, brush)
brush = QBrush(QColor(255, 255, 255))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Disabled, QPalette.Window, brush)
brush = QBrush(QColor(0, 0, 0))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Disabled, QPalette.Shadow, brush)
brush = QBrush(QColor(255, 255, 255))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Disabled, QPalette.AlternateBase, brush)
brush = QBrush(QColor(255, 255, 255))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Disabled, QPalette.ToolTipBase, brush)
brush = QBrush(QColor(0, 0, 0))
brush.setStyle(Qt.SolidPattern)
palette.setBrush(QPalette.Disabled, QPalette.ToolTipText, brush)
self.runtimeTerminal_label.setPalette(palette)
self.runtimeTerminal_label.setMouseTracking(True)
self.runtimeTerminal_label.setAutoFillBackground(False)
self.runtimeTerminal_label.setFrameShape(QFrame.StyledPanel)
self.runtimeTerminal_label.setFrameShadow(QFrame.Sunken)
self.runtimeTerminal_label.setText("")
self.runtimeTerminal_label.setObjectName("runtimeTerminal_label")
self.verticalLayout_10.addWidget(self.runtimeTerminal_label)
self.verticalLayout_12 = QVBoxLayout()
self.verticalLayout_12.setContentsMargins(-1, 0, -1, -1)
self.verticalLayout_12.setObjectName("verticalLayout_12")
self.transferPNRDSetup_pushButton = QPushButton(self.tab_petrinet)

self.transferPNRDSetup_pushButton.setObjectName("transferPNRDSetup_pushButton")
self.verticalLayout_12.addWidget(self.transferPNRDSetup_pushButton)
self.getRuntimeInfo_pushButton = QPushButton(self.tab_petrinet)
self.getRuntimeInfo_pushButton.setMinimumSize(QSize(0, 30))
self.getRuntimeInfo_pushButton.setMaximumSize(QSize(9999999, 16777215))

```

```

self.getRuntimeInfo_pushButton.setObjectName("getRuntimeInfo_pushButton")
self.verticalLayout_12.addWidget(self.getRuntimeInfo_pushButton)
self.generateNewPNML_pushButton = QPushButton(self.tab_petrinet)

self.generateNewPNML_pushButton.setObjectName("generateNewPNML_pushButton")
self.verticalLayout_12.addWidget(self.generateNewPNML_pushButton)
self.verticalLayout_10.addLayout(self.verticalLayout_12)
self.setup_tabWidget.addTab(self.tab_petrinet, "")
self.verticalLayout_11.addWidget(self.setup_tabWidget)
MainWindow.setCentralWidget(self.centralwidget)
self.menubar = QMenuBar(MainWindow)
self.menubar.setGeometry(QRect(0, 0, 931, 22))
self.menubar.setObjectName("menubar")
self.menuFile = QMenu(self.menubar)
self.menuFile.setObjectName("menuFile")
MainWindow.setMenuBar(self.menubar)
self.actionopen_pnml = QAction(MainWindow)
icon = QIcon()
icon.addPixmap(QPixmap(resource_path("images/xml.svg")), QIcon.Normal,
QIcon.Off)
self.actionopen_pnml.setIcon(icon)
self.actionopen_pnml.setObjectName("actionopen_pnml")
self.actionExit = QAction(MainWindow)
self.actionExit.setObjectName("actionExit")
self.actionOpen_Setup_File_pelms = QAction(MainWindow)
self.actionOpen_Setup_File_pelms.setObjectName("actionOpen_Setup_File_pelms")
self.menuFile.addSeparator()
self.menuFile.addAction(self.actionopen_pnml)
self.menuFile.addSeparator()
self.menuFile.addAction(self.actionOpen_Setup_File_pelms)
self.menuFile.addSeparator()
self.menuFile.addSeparator()
self.menuFile.addAction(self.actionExit)
self.menubar.addAction(self.menuFile.menuAction())

self.retranslateUi(MainWindow)
self.setup_tabWidget.setCurrentIndex(0)
self.actionExit.triggered.connect(self.closeEvent)
QMetaObject.connectSlotsByName(MainWindow)

def closeEvent(self, event):
    msgBox = QMessageBox()
    msgBox.setIcon(QMessageBox.Icon.Question)
    msgBox.setText("Are you sure you want to close the application?")
    msgBox.setWindowTitle("Confirmation")
    msgBox.setStandardButtons(QMessageBox.Yes | QMessageBox.No)
    reply = msgBox.exec()
    #reply = QMessageBox.question('Confirmation', 'Are you sure you want to close the
application?', [QMessageBox.Yes | QMessageBox.No], QMessageBox.No)

```

```

if reply == QMessageBox.Yes:
    if not type(event) == bool:
        event.accept()
    else:
        sys.exit()
else:
    if not type(event) == bool:
        event.ignore()

def retranslateUi(self, MainWindow):
    _translate = QCoreApplication.translate

    #----- Window Title Text -----
    MainWindow.setWindowTitle(_translate("MainWindow", "PELMS-MQTT"))

    #----- Setup Tab Text -----
    self.label_6.setText(_translate("MainWindow", "Setup PNRD\n Type"))
    self.setupPelms_comboBox.setItemText(0, _translate("MainWindow", "PNRD"))
    self.setupPelms_comboBox.setItemText(1, _translate("MainWindow", "iPNRD"))
    self.label_7.setText(_translate("MainWindow", "Reader Name: "))
    self.label_8.setText(_translate("MainWindow", "Number of Antennas"))
    self.label_3.setText(_translate("MainWindow", "Platform IP"))
    self.IP_lineEdit.setInputMask(_translate("MainWindow", "000.000.000.000;_"))
    self.IP_lineEdit.setText(_translate("MainWindow", "..."))
    self.addIP_pushButton.setText(_translate("MainWindow", "Add"))
    self.createSetup_pushButton.setText(_translate("MainWindow", "CREATE SETUP
FILE"))
    self.marking_vector.setText(_translate("MainWindow", "Marking Vector"))
    self.label.setText(_translate("MainWindow", "Incidence Matrix"))
    self.label_4.setText(_translate("MainWindow", "Incidence Matrix Transpose Array"))
    self.label_2.setText(_translate("MainWindow", "Marking Array"))
    self.setup_tabWidget.setTabText(self.setup_tabWidget.indexOf(self.tab_setup),
_translate("MainWindow", "SETUP"))

    #-----
    self.setupInfo_label.setText(_translate("MainWindow", "<html><head/><body><p
align=\"center\"><br/></p></body></html>"))
    self.readerList_label.setText(_translate("MainWindow", "<html><head/><body><p
align=\"center\"><br/></p></body></html>"))
    self.places_label.setText(_translate("MainWindow",
"<html><head/><body><p><br/></p></body></html>"))
    self.transitions_label.setText(_translate("MainWindow",
"<html><head/><body><p><br/></p></body></html>"))

    #----- Runtime Tab Text -----
    self.label_9.setText(_translate("MainWindow", "Marking Vector"))
    self.label_10.setText(_translate("MainWindow", "Incidence Matrix"))
    self.transferPNRDSetup_pushButton.setText(_translate("MainWindow",
"TRANSFER PNRD SETUP"))

```

```

        self.getRuntimeInfo_pushButton.setText(_translate("MainWindow", "GET RUNTIME
INFO"))
        self.generateNewPNML_pushButton.setText(_translate("MainWindow",
"TRANSFER INITIAL DATA"))
        self.setup_tabWidget.setTabText(self.setup_tabWidget.indexOf(self.tab_petrinet),
_translate("MainWindow", "RUNTIME"))

#----- File Menu Text -----
self.menuFile.setTitle(_translate("MainWindow", "File"))
self.actionopen_pnml.setText(_translate("MainWindow", "Open PNML file"))
self.actionopen_pnml.setShortcut(_translate("MainWindow", "Ctrl+O"))
self.actionOpen_Setup_File_pelms.setText(_translate("MainWindow", "Open  Setup
File (*.pelms)"))
self.actionOpen_Setup_File_pelms.setShortcut(_translate("MainWindow", "Ctrl+F"))
self.actionExit.setText(_translate("MainWindow", "Exit"))
self.actionExit.setShortcut(_translate("MainWindow", "Ctrl+X"))

if __name__ == "__main__":
    app = QApplication(sys.argv)
    MainWindow = QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())

```

APÊNDICE D PELMSGUI.PY

```
#!/usr/bin/python3
'''
Created on Nov, 2021
Updated on:
    * Jan, 2022
    * Dec, 2024
@author: Roger Henrique Carrijo de Paula
@updater author:
    * Daniel Barbosa Pereira
    * Álisson Carvalho Vasconcelos
@Github authors:
    * rogercarrijo
    * daniel-b-pereira
    * AlissonCV
'''

# Import System Class
import os
import sys

# Import MQTT Json Class
import json

# Import Time Class
from datetime import datetime as dt

# Importing Pyqt5 Class
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

# Import GUI Class
from gui.MainWindow import Ui_MainWindow
from gui.PopupInfo import PopupInfo

from pelms.Pnrd import Pnrd
from pelms.FileCreator import FileCreator
from pelms.FileCopy import FileCopy
from pelms.utils.template import *

from paho.mqtt import client as mqtt_client

class PelmsGui():
    # def __init__
    # def connect_MQTT
    # def PNRD_setup
    # def open_pnml_file
```

```

# def PNRD_pelms_runtime
# def open_pelms_file
# def create_PELMS_file
# def setup_PELMS_type
# def append_reader
# def get_transition_names
# def set_antennas
# def setup_matrix_view
# def setup_matrix_vector
# def setup_marking_vector
# def publish_mqtt
# def transferPNRD
# def publish_inicial
# def receive_mqtt
# def mqtt_fire
def __init__(self):
    self.pathName = "
#     self.MQTT_array = {}

#----- PELMS MQTT Variables -----
-----
#----- MQTT Variables -----
-----

self.brokerName = 'Mosquitto'
self.broker = 'test.mosquitto.org'
self.brokerName = 'MAPL'
self.broker = '200.19.144.16'
self.brokerName = 'RPi-MAPL'
self.broker = '192.168.1.13'
self.brokerName = 'RPi-MAPL'
self.broker = '192.168.12.9'
self.port = 1883
self.username = 'XXXX'
self.password = 'XXXX'

self.client_id = 'PELMS'
self.topic = {
    'place': 'PELMS-MQTT/len_place',
    'transition': 'PELMS-MQTT/len_transition',
    'token': 'PELMS-MQTT/marketing_vector',
    'incidence': 'PELMS-MQTT/incidence_matrix',
    'fire': 'PELMS-MQTT/fire_vector',
    'runtime': 'PELMS-MQTT/Runtime/'
}

self.client = mqtt_client.Client(self.client_id)
self.mqtt_runtime_flag = 'OFF'
self.pnrd_mqtt = dict()

```

```

#----- PNRD Variables -----
-----
self.pnrd = Pnrd()
self.pnrd_setup_is_ok = False
self.pelms_type = ""

#----- Petri Net Variables -----
-----
self.count_antenna = 0
self.qtd_antena = 0
self.reader_list = []
self.transition_names = []
self.array_matrix = []
self.array_marking = []
self.starting_token_vector = []

#----- PELMS MainWindow Implementation -----
-----
#----- MainWindow Creation -----
-----
app = QApplication(sys.argv)
self.MainWindow = QMainWindow()
self.ui = Ui_MainWindow()
self.ui.setupUi(self.MainWindow)

#----- File Open Menu -----
-----
self.ui.actionopen_pnml.triggered.connect(self.open_pnml_file)
self.ui.actionOpen_Setup_File_pelms.triggered.connect(self.open_pelms_file)

#----- Setup PNRD Type List -----
-----
self.ui.setupPelms_comboBox.currentIndexChanged.connect(self.setup_PELMS_type)

#----- Add Button -----
-----
self.ui.addIP_pushButton.clicked.connect(self.append_reader)

#----- Export PELMS Setup Button -----
-----
self.ui.createSetup_pushButton.clicked.connect(self.create_PELMS_file)

#----- Setup Antenna Input Box -----
-----
self.ui.nAntennas_spinBox.setMinimum(1)
self.ui.nAntennas_spinBox.setMaximum(3)
self.ui.nAntennas_spinBox.setValue(1)

#----- Transfer PNRD Setup Button -----
-----

```

```

self.ui.transferPNRDSetup_pushButton.clicked.connect(self.transferPNRD)

#----- Get Runtime Info Button -----
-----
self.ui.getRuntimeInfo_pushButton.clicked.connect(self.receive_mqtt)

#----- Transfer Initial Data Button -----
-----
self.ui.generateNewPNML_pushButton.clicked.connect(self.publish_inicial)

#----- Display PELMS MQTT Window -----
-----
self.MainWindow.show()
sys.exit(app.exec_())

#----- Connect to MQTT Server -----
-----
def connect_MQTT(self):
    #----- Connect Event -----
    -----
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print(f'Connected to {self.brokerName} Broker!')
            client.subscribe(f'{self.topic["runtime"]}')
        else:
            print(f'Failed to connect, return code %d\n', rc)

    #----- Message Event -----
    -----
    def on_message(client, userdata, msg):
        if self.mqtt_runtime_flag == "ON":
            print(f'Recebido \"{msg.payload.decode()}\" de \"{msg.topic}\" topic')

            if f'{self.topic["runtime"]}' == msg.topic:
                self.pnr_d_mqtt['id'] = msg.payload.decode()
                self.ui.id_label.setText("TagId: "+str( self.pnr_d_mqtt['id']))
            elif f'{self.topic["runtime"]}' == msg.topic:
                self.pnr_d_mqtt['reader'] = msg.payload.decode()
                self.ui.reader_label.setText("Reader: "+str( self.pnr_d_mqtt['reader']))
            elif f'{self.topic["runtime"]}' == msg.topic:
                self.pnr_d_mqtt['error'] = msg.payload.decode()
                self.ui.exception_label.setText("PNRD: "+str( self.pnr_d_mqtt['error']))
            elif f'{self.topic["runtime"]}' == msg.topic:
                self.pnr_d_mqtt['antenna'] = msg.payload.decode()
            elif f'{self.topic["runtime"]}' == msg.topic:
                self.pnr_d_mqtt['fire'] = msg.payload.decode()
                self.pnr_d_mqtt['date'] = dt.now().strftime("%m/%d/%Y")
                self.pnr_d_mqtt['time'] = dt.now().strftime("%H:%M:%S")

```

```

        self.ui.incMatrix2_tw.item(self.count_col,
self.pnrd.len_places).setBackground(QColor('White'))
        self.count_col = int(self.pnrd_mqtt['fire'])
        if self.pnrd_mqtt['error'].lower() == "no_error":
            self.ui.incMatrix2_tw.item(self.count_col,
self.pnrd.len_places).setBackground(QColor('LightGreen'))
            self.mqtt_fire(self.pnrd_mqtt['fire'])
        elif self.pnrd_mqtt['error'].lower() == "exception":
            self.ui.incMatrix2_tw.item(self.count_col,
self.pnrd.len_places).setBackground(QColor('Tomato'))
        else:
            self.ui.incMatrix2_tw.item(self.count_col,
self.pnrd.len_places).setBackground(QColor('Gold'))

        #print(self.pnrd_mqtt)
        runtime = FileCreator('pelmsSetup/runtime', self.pathName, 'runtime', 'pnrd')
        runtime.set_text_increment(
            template_mqtt(
                tagid = self.pnrd_mqtt['id'],
                reader = self.pnrd_mqtt['reader'],
                error = self.pnrd_mqtt['error'],
                antenna = self.pnrd_mqtt['antenna'],
                fire = self.pnrd_mqtt['fire'],
                date = self.pnrd_mqtt['date'],
                time = self.pnrd_mqtt['time']
            )
        )

#----- Server MQTT Variable -----
-----
if self.username != 'XXXX' and self.password != 'XXXX':
    self.client.username_pw_set(self.username, self.password)
self.client.on_connect = on_connect
self.client.on_message = on_message
self.client.connect(self.broker, self.port)
self.client.loop_start()

#----- Open PNML File -----
-----
#----- Run Petri Net and call for display -----
-----
def PNRD_setup(self, fileName):
    print(self.pathName+fileName)
    _,success = self.pnrd.set_pnml(self.pathName + fileName)
    if success:
        _,success = self.pnrd.create_net()
        if success:
            self.setup_matrix_view(self.pnrd.len_transitions,self.pnrd.len_places)
            self.setup_matrix_vector()
            self.setup_marking_vector(self.pnrd.len_places)

```

```

self.pnrd_setup_is_ok = True
self.count_antenna = self.pnrd.len_transitions
self.ui.nAntennas_spinBox.setRange(1, 3)
self.ui.qtdTotalTansitions_label.setText(f'(Left: {self.count_antenna})')
self.setup_PELMS_type()

#----- Run Open PNML File -----
-----

def open_pnml_file(self):
    self.ui.createSetup_pushButton.setEnabled(True)

    self.ui.addIP_pushButton.setEnabled(True)

    self.fileName, _ = QFileDialog.getOpenFileName(filter="PNML (*.pnml)")
    self.pathName = os.path.dirname(os.path.abspath(self.fileName))
    self.fileName = (self.fileName.split('/'))[-1].replace('.pnml', '')

    self.ui.setup_tabWidget.setCurrentIndex(0)
    self.ui.setup_tabWidget.setTabEnabled(0, True)
    self.ui.setup_tabWidget.setTabEnabled(1, False)

    self.ui.readerName_lineEdit.setText("")
    self.ui.qtdTotalTansitions_label.setText('(Left: 0)')
    self.ui.nAntennas_spinBox.setRange(0, 0)
    self.ui.IP_lineEdit.setText("")
    self.ui.setupInfo_label.setText("")
    self.ui.markingVector_tw.setRowCount(0)
    self.ui.markingVector_tw.setColumnCount(0)
    self.ui.incMatrix_tw.setRowCount(0)
    self.ui.incMatrix_tw.setColumnCount(0)
    self.ui.matrix_array.setText("")
    self.ui.marking_array.setText("")
    self.text_setup = ""
    self.reader_list = []

    if self.fileName != "":
        self.PNRD_setup(f'/{self.fileName}.pnml')

#----- Open PELMS File -----
-----

#----- Run Petri Net and call for display -----
-----

def PNRD_pelms_runtime(self, fileName):
    with open(self.pathName + fileName, 'r') as pelms_file:
        pelms = json.load(pelms_file)
        _, success = self.pnrd.set_pnml(pelms["pnmlFile"])
        if success:
            _, success = self.pnrd.create_net()
            if success:
                self.pnrd.transition_names = pelms["transitionNames"]

```

```

        self.setup_matrix_view(self.pnrd.len_transitions,self.pnrd.len_places,"pelms")
        self.setup_matrix_vector("pelms")
        self.setup_marking_vector(self.pnrd.len_places,"pelms")
        self.pnrd_setup_is_ok = True
        self.setup_PELMS_type()
        self.ui.pelmsType_label.setText(f'Type: {pelms["type"]}')
        self.ui.qtdReader_label.setText(f'Qtd Readers: {pelms["qtdReaders"]}')
        readers_list = ""
        for i in pelms["readerListConfig"]:
            readers_list += f'Reader: {i["readerName"]} \n Qtd Ant: {i["qtdAntenna"]} \n
IP: {i["IP"]} \n \n'

        self.ui.readerList_label.setText(readers_list)
        self.reader_list = pelms["readerListConfig"]

#----- Run Open PELMS File -----
def open_pelms_file(self):
    self.fileName, _ = QFileDialog.getOpenFileName(filter="PELMS (*.pelms)")
    self.pathName = (os.path.dirname(os.path.abspath(self.fileName))).replace("/pelmsSetup", "")
    fileName = self.fileName.replace(self.pathName, "")
    self.fileName = self.pathName.split('/')[1]

    self.ui.setup_tabWidget.setCurrentIndex(1)
    self.ui.setup_tabWidget.setTabEnabled(1, True)
    self.ui.setup_tabWidget.setTabEnabled(0, False)

    if fileName != "":
        self.PNRD_pelms_runtime(fileName)
        self.connect_MQTT()
        self.count_col = 0

#----- Create PELMS Files -----
# TODO
def create_PELMS_file(self):
    path = (f'{QFileDialog.getSaveFileName(directory = self.fileName, filter="All Files (*.*)", ) [0]}').replace(f'{self.fileName}/',) if 'PNMLexamples' in self.pathName else self.pathName
    path = path if self.fileName in path else f'{path}/{self.fileName}'
    self.pelms_type = self.ui.setupPelms_comboBox.currentText()
    self.get_transition_names(self.pnrd.len_transitions,self.pnrd.len_places)

    pelms_file = FileCopy(path, self.pathName, self.fileName, self.fileName, 'pnml')
    pelms_file.copy_file() if 'PNMLexamples' in self.pathName else pelms_file.move_file()
    pelms_file = FileCreator('pelmsSetup', path,'setup','pelms')
    dict_pelms = {
        "pnmlFile": self.pnrd.file,

```

```

        "type": self.pelms_type,
        "qtdReaders": len(self.reader_list),
        "readerListConfig": self.reader_list,
        "transitionNames": self.transition_names
    }
    pelms_file.set_text(json.dumps(dict_pelms, indent=4, sort_keys=True))

    fileName = f'{self.pelms_type.lower()}Data'
    pnr_data = FileCreator('pelmsSetup', path, fileName, 'pnr')
    pnr_data.set_text(
        template_pnr_data(
            n_places = self.pnr.len_places,
            n_transitions = self.pnr.len_transitions,
            IncidenceMatrix = ','.join(map(str, self.array_matrix)),
            StartingTokenVector = ','.join(map(str, self.starting_token_vector)),
            FireVector = None if self.pelms_type.lower() == 'PNRD' else ('1' if i == 0 else '0'
for i in range(self.pnr.len_transitions))
        )
    )

    fileName = f'{self.pelms_type.lower()}_recorder'
    reader_ino = FileCreator(f'pelmsSetup/ino/{fileName}', path, fileName, 'ino')
    reader_ino.set_text(template_recorder(self.pelms_type, self.topic))
    wifi_h = FileCreator(f'pelmsSetup/ino/{fileName}', path, 'WiFiPassword', 'h')
    wifi_h.set_text(
        template_recorder_h(
            'TROJANUBL',
            'CABRAL12',
            'ESP32-RECORDER',
            self.broker,
            1883,
            1,
            self.topic['runtime'].replace('Runtime/', '#')
        )
    )

    i = 0
    antenna_id_list = ""
    reader_list = list()

    for reader in self.reader_list:
        fileName = f'{self.pelms_type.lower()}_reader_{i}' if len(self.reader_list) > 1 else
f'{self.pelms_type.lower()}_reader'
        reader_ino = FileCreator(f'pelmsSetup/ino/{fileName}', path, fileName, 'ino')
        reader_ino.set_text(template_reader(self.pelms_type, self.topic))
        wifi_h = FileCreator(f'pelmsSetup/ino/{fileName}', path, 'WiFiPassword', 'h')

        reader_list.append(f'pelms {i}') if reader["readerName"] == " " else
reader_list.append(reader["readerName"])
        antenna_list = list()

```

```

position_list = list()
for count in range(reader["qtdAntenna"]):
    antenna_list.append(count)
    position_list.append(i + count)
antenna_id_list += f'{" " if i == 0 and i < len(reader) else ","} {antenna_list}'

wifi_h.set_text(
    template_reader_h(
        'TROJANUBL',
        'CABRAL12',
        'ESP32-RECORDER',
        self.broker,
        1883,
        1,
        self.topic['runtime'].replace('Runtime/', '#'),
        reader_list[i],
        ",".join(map(str, antenna_list)),
        i
    )
)

i += 1

self.ui.popup_info = PopupInfo("Successfully created PELMS file!\n\nTo open file and
use runtime mode press (Ctrl + F)")
self.ui.popup_info.show()
self.ui.createSetup_pushButton.setEnabled(False)

self.ui.addIP_pushButton.setEnabled(False)
self.text_setup = "

#----- Setup PNRD Type -----
-----
def setup_PELMS_type(self):
    self.pelms_type = self.ui.setupPelms_comboBox.currentText()
    if self.pelms_type=='iPNRD':
        self.ui.nAntennas_spinBox.setMinimum(1)
        self.ui.nAntennas_spinBox.setMaximum(1)
    if self.pelms_type=='PNRD':
        self.ui.nAntennas_spinBox.setMinimum(1)
        self.ui.nAntennas_spinBox.setMaximum(3)

    self.ui.qtdTotalTansitions_label.setText(f'(Left: {self.count_antenna})')

    return self.pelms_type

#----- Display Create Setup File Menu Text -----
-----
def append_reader(self):
    pelms_type = self.setup_PELMS_type()

```

```

reader_name = self.ui.readerName_lineEdit.text()

#- TODO: config that the same IP cannot be used twice -#
ip_connection = self.ui.IP_lineEdit.text() if self.ui.IP_lineEdit.text() != self.broker else
f'{self.broker.replace(self.broker[-1], str(int(self.broker[-1]) + 1))}'
self.ui.IP_lineEdit.setText(ip_connection)

if pelms_type == 'PNRD':
    self.set_antennas()
#    print(f'Antenna Used: {self.qtd_antena}\tAntenna Left: {self.count_antenna}')

self.reader_list.append({"readerName":reader_name,"qtdAntenna":self.qtd_antena,"IP":ip_
connection})

    self.text_setup += f'\nReader: {reader_name} IP: '{ip_connection}' Ant:
{self.qtd_antena} units'
    else:
        self.set_antennas()
#        print(f'Antenna Used: {self.qtd_antena}\tAntenna Left: {self.count_antenna}')

self.reader_list.append({"readerName":reader_name,"qtdAntenna":1,"IP":ip_connection})

    self.text_setup += f'\nReader: {reader_name} IP: '{ip_connection}' Ant: 1 unit'

    self.ui.setupInfo_label.setText(f'P: {self.pnrd.len_places} | T:
{self.pnrd.len_transitions} {self.text_setup}')

#----- Get Transition Names Displayed -----
def get_transition_names(self,n_row,n_col):
    self.transition_names = []
    for i in range(n_row):
        matrix_transition_item = self.ui.incMatrix_tw.item(i,n_col)
        self.transition_names.append(matrix_transition_item.text())

#----- Count and Limit Qtd of Antenna -----
def set_antennas(self):
    if self.count_antenna >= self.ui.nAntennas_spinBox.value():
        self.count_antenna -=self.ui.nAntennas_spinBox.value()
        self.qtd_antena = self.ui.nAntennas_spinBox.value()

    if self.count_antenna > 0:
        self.setup_PELMS_type()
    elif self.count_antenna == 0:
        self.ui.nAntennas_spinBox.setMaximum(0)
        self.ui.nAntennas_spinBox.setMinimum(0)
        self.ui.qtdTotalTansitions_label.setText(f'(Left: {self.count_antenna}'))

```

```

self.ui.addIP_pushButton.setEnabled(False)

#----- Display the Incidence Matrix Table -----
-----
def setup_matrix_view(self,n_row,n_col,_type="pnml"):
    self.ui.incMatrix2_tw.setRowCount(n_row) if _type == "pelms" else
self.ui.incMatrix_tw.setRowCount(n_row)
    self.ui.incMatrix2_tw.setColumnCount(n_col+1) if _type == "pelms" else
self.ui.incMatrix_tw.setColumnCount(n_col+1)

    count_row = 0
    horizontalHeader = []
    verticalHeader = []

    for row in self.pnrd.incidence_matrix:
        count_col = 0
        for i in row:
            if len(horizontalHeader) < n_col:
                horizontalHeader.append(f" P{count_col} ")

                if count_col==(n_col -1):
                    self.ui.incMatrix2_tw.setItem(
count_row,count_col+1,
QTableWidgetItem(f'{self.pnrd.transition_names[count_row]}')) if _type == "pelms" else
self.ui.incMatrix_tw.setItem(
count_row,count_col+1, QTableWidgetItem(f'transition
{count_row}'))

                    self.ui.incMatrix2_tw.setItem(
count_row,count_col,
QTableWidgetItem(f'{i}')) if _type == "pelms" else self.ui.incMatrix_tw.setItem(
count_row,count_col, QTableWidgetItem(f'{i}'))

                count_col+=1

            verticalHeader.append(f" T{count_row} ")
            count_row += 1

        horizontalHeader.append(f" ")

        self.ui.incMatrix2_tw.setHorizontalHeaderLabels(horizontalHeader) if _type ==
"pelms" else self.ui.incMatrix_tw.setHorizontalHeaderLabels(horizontalHeader)
        self.ui.incMatrix2_tw.setVerticalHeaderLabels(verticalHeader) if _type == "pelms"
else self.ui.incMatrix_tw.setVerticalHeaderLabels(verticalHeader)

        self.ui.places_label.setText(f'Places: {self.pnrd.len_places}') if _type == "pelms" else
None
        self.ui.transitions_label.setText(f'Transitions: {self.pnrd.len_transitions}') if _type ==
"pelms" else None

#----- Display the Incidence Matrix Transpose Array -----
-----
def setup_matrix_vector(self, type="pnml"):

```

```

self.array_matrix = []

for row in self.pnrd.incidence_matrix_t:
    for pos in row:
        self.array_matrix.append(pos)

None if _type == "pelms" else self.ui.matrix_array.setText(f'{self.array_matrix}')

#----- Display the Marking Vector Table -----
#-----
def setup_marking_vector(self,n_row,_type="pnml"):
    self.ui.markingVector2_tw.setRowCount(n_row) if _type == "pelms" else
self.ui.markingVector_tw.setRowCount(n_row)
    self.ui.markingVector2_tw.setColumnCount(1) if _type == "pelms" else
self.ui.markingVector_tw.setColumnCount(1)

    count_row = 0
    verticalHeader = []
    self.array_marking= []
    self.starting_token_vector = []

    for i in self.pnrd.marking_vector:
        # TODO: Confirm if the starting token vector in pnml is the first position
        self.starting_token_vector.append(i)

        verticalHeader.append(f' P{count_row} ')

        self.ui.markingVector2_tw.setItem(count_row,0, QTableWidgetItem(f'{i}')) if
_type == "pelms" else self.ui.markingVector_tw.setItem(count_row,0,
QTableWidgetItem(f'{i}'))
        if 1 == i and _type == "pelms":
            self.ui.markingVector2_tw.item(count_row,
0).setBackground(QColor('LightGreen'))

        self.array_marking.append(i)
        count_row+=1

    self.ui.markingVector2_tw.setHorizontalHeaderLabels([""]) if _type == "pelms" else
self.ui.markingVector_tw.setHorizontalHeaderLabels([""])
    self.ui.markingVector2_tw.setVerticalHeaderLabels(verticalHeader) if _type ==
"pelms" else self.ui.markingVector_tw.setVerticalHeaderLabels(verticalHeader)

#----- Display the Marking Vector Array -----
#-----
None if _type == "pelms" else self.ui.marking_array.setText(f'{self.array_marking}')

#----- MQTT Publishing PELMS DATA -----
#-----

```

```

#----- Call Publish PNRD to Reader -----
-----
def transferPNRD(self):
    self.publish_mqtt(f'pelmsSetup/{self.pelms_type.lower()}Data.pnrd', True)

#----- Call Publish PNRD to Recorder -----
-----
def publish_inicial(self):
    self.publish_mqtt(f'pelmsSetup/{self.pelms_type.lower()}Data.pnrd')

#----- Publish PNRD Message -----
-----
def publish_mqtt(self, fileName, type = False):
    with open(self.pathName + f'/{fileName}', 'r') as mqtt:
        file_text = mqtt.read().splitlines()
        topic = ""

        for i in range(len(file_text)):
            for j in self.topic.keys():
                if j in file_text[i]:
                    text_mqtt = file_text[i].replace(f'{j}: ', '')
                    #print(f'{j}, {text_mqtt}')

                    if type:
                        topic = self.topic[j].split('/')
                        topic = self.topic[j].replace(topic[-1], f'fire/{topic[-1]}')
                    else:
                        topic = self.topic[j]

                    result = self.client.publish(topic, text_mqtt)

                    if not result[0]:
                        print(f'Sucess to send message to topic {topic}')
                    else:
                        print(f'Failed to send message to topic {topic}')

            mqtt.close()

#----- MQTT Receive PELMS Runtime Data -----
-----
def receive_mqtt(self):
    self.mqtt_runtime_flag = 'ON'

#----- Fire Petri Nets View -----
-----
def mqtt_fire(self, fire_position):
    fire_position = fire_position.replace(f'{fire_position}', f'transition {fire_position}')

    count_row = 0
    for row in range(self.pnrd.len_transitions):

```

```

matrix_transition_item = self.ui.incMatrix2_tw.item(row, self.pnrd.len_places)
if fire_position == matrix_transition_item.text():
    break
count_row += 1

count_col = [0, 0]
flag = [False, False]
for col in self.pnrd.incidence_matrix[count_row]:
    if abs(col) != 1:
        count_col[0] += 1 if not flag[0] else 0
        count_col[1] += 1 if not flag[1] else 0
    elif col == -1:
        flag[0] = True
        count_col[1] += 1 if not flag[1] else 0
    elif col == 1:
        flag[1] = True
        count_col[0] += 1 if not flag[0] else 0

    if flag[0] and flag[1]:
        break

self.ui.markingVector2_tw.setItem(count_col[0], 0, QTableWidgetItem("0"))
self.ui.markingVector2_tw.setItem(count_col[1], 0, QTableWidgetItem("1"))
self.ui.markingVector2_tw.item(count_col[0], 0).setBackgroundColor(QColor('White'))
self.ui.markingVector2_tw.item(count_col[1],
0).setBackgroundColor(QColor('LightGreen'))

#----- Run if the file is the main -----
-----
if __name__ == "__main__":
    PelmsGui()

```

APÊNDICE E TEMPLATE.PY

```

"""
Created on Nov, 2021
Updated on:
    * Jan, 2022
    * Jan, 2025
@author: Daniel Barbosa Pereira
@updater author:
    * Roger Henrique Carrijo de Paula
    * Daniel Barbosa Pereira
    * Álisson Carvalho Vasconcelos
@Github authors:
    * rogercarrijo
    * daniel-b-pereira
    * AlissonCV
"""

import os

def template_mqtt(tagid, reader, error, antenna, fire, date, time):
    return f'\
Tag Id: {tagid}\n\
Reader: {reader}\n\
Error: {error}\n\
Antenna: {antenna}\n\
Fire Position: {fire}\n\
Date: {date}\n\
Time: {time}\n\
\n\n'

def template_pnrd_data(n_places, n_transitions, IncidenceMatrix, StartingTokenVector,
FireVector):
    text_format = f'place: {n_places}\ntransition: {n_transitions}\ntoken:
{{{StartingTokenVector}}}\nincidence: {{{IncidenceMatrix}}}'

    if FireVector == None:
        text_format += f'\nfire: {{{FireVector}}}'

    return text_format

def template_recorder(pelmsType, topic):
    pelmsType = pelmsType.lower()
    pathName = f'{os.getcwd()}/pelms/utils'
    finalText = "

    with open(f'{pathName}/template_begin.ino', "r") as begin:
        with open(f'{pathName}/template_recorder_{pelmsType}.ino') as pnrd:
            for text in begin:

```

```

        finalText += text
    finalText += "\n\n"
    for text in pnrdr:
        finalText += text

if pelmsType == 'pnrd':
    pnrdr_dict = {
        'place': 1,
        'transition': 2,
        'token': 3,
        'incidence': 4
    }
else:
    pnrdr_dict = {
        'place': 1,
        'transition': 2,
        'fire': 3
    }

for i in pnrdr_dict.keys():
    finalText = finalText.replace(fPELMSVAR{pnrdr_dict[i]}', topic[i])

return finalText

def template_reader(pelmsType, topic):
    pelmsType = pelmsType.lower()
    pathName = f'{os.getcwd()}/pelms/utils'
    finalText = ""

    with open(f'{pathName}/template_begin.ino', "r") as begin:
        with open(f'{pathName}/template_reader_{pelmsType}.ino') as pnrdr:
            for text in begin:
                finalText += text
            finalText += "\n\n"
            for text in pnrdr:
                finalText += text

if pelmsType == 'pnrd':
    pnrdr_dict = {
        'place': 1,
        'transition': 2,
        'runtime': 3
    }
else:
    pnrdr_dict = {
        'place': 1,
        'transition': 2,
        'incidence': 3,
        'token': 4,
        'runtime': 5
    }

```

```

    }

runtime_dict = {
    'id': 0,
    'reader': 1,
    'error': 2,
    'antenna': 3,
    'fire': 4
}

for i in pnr_dict.keys():
    if i == 'runtime':
        for j in runtime_dict.keys():
            finalText = finalText.replace(f'PELMSVAR{pnr_dict[i]+runtime_dict[j]}',
f'{topic[i]}{j}')
        else:
            topic_array = topic[i].split('/')
            finalText = finalText.replace(f'PELMSVAR{pnr_dict[i]}',
topic[i].replace(topic_array[-1], f'fire/{topic_array[-1]}'))

    return finalText

def template_recorder_h(ssid, passWord, clientid, server, port, serial, topic,
userName=None, anonymous=None):
    return template_wifi_h(ssid, userName, passWord, anonymous, clientid, server, port,
serial, topic)

def template_reader_h(ssid, passWord, clientid, server, port, serial, topic, reader, antennaid,
position, userName=None, anonymous=None):
    return template_wifi_h(ssid, userName, passWord, anonymous, clientid, server, port,
serial, topic, reader, antennaid, position)

def template_wifi_h(ssid, userName, passWord, anonymous, clientid, server, port, serial,
topic, readerid=None, antennaid=None, position=None):
    pathName = f'{os.getcwd()}/pelms/utils'
    finalText = "

    pelms_dict = {
        'ssid': ssid,
        'username': 'XXXX' if userName == None else userName,
        'password': passWord,
        'anonymous': 'anonymous@ufu.br' if anonymous == None else anonymous,
        'clientid': clientid,
        'server': server,
        'port': port,
        'serial': serial,
        'topic': topic,
        'readerid': readerid,
        'antennaid': antennaid,
        'position': position
    }

```

```

}

with open(f'{pathName}/template_wifipassword.h', "r") as file:
    for text in file:
        finalText += text

if pelms_dict['readerid'] != None:
    finalText = f'{finalText}\n\n/* DECLARATION OF FILE VARIABLES */\nString
pelmsReader = \"PELMSVAR10\";\nString pelmsAntenna = \"PELMSVAR11\";\nString
pelmsPosition = \"PELMSVAR12\";'

j = 0
for i in pelms_dict.keys():
    j += 1
    if j < 10:
        finalText = finalText.replace(fPELMSVAR0{j}', f'{pelms_dict[i]}')
    else:
        finalText = finalText.replace(fPELMSVAR{j}', f'{pelms_dict[i]}')

return finalText

```

APÊNDICE F TEMPLATE_BEGIN.INO

```

/**
 * Created on Dec, 2024
 * @headerfile: PNRD recorder
 *
 * @details: This code receive message from MQTT protocol and set the variables of
 * the PNRD/iPNRD to be stored in a tag.
 * 1- MQTT set the variables of PNRD/iPNRD
 * 2- When tag presented it erase, restore to factory and format the tag as NDEF format
 * 3- Then it stores the PNRD/iPNRD in the tag using NDEF format
 *
 * @author: Álisson Carvalho Vasconcelos
 * @authors updater:
 * @authors github: AlissonCV
 *
 * @copyright: Free for All
 */

#include <EWiFi.h>
#include <PubSubClient.h>
#include <WiFiPassword.h>
#include <HardwareSerial.h>
#include <Pn532NfcReader.h>

/* DECLARATION OF OBJECT FOR MQTT COMUNICATION AND LCD
DISPLAYING */
WiFiClient espClient;
PubSubClient MQTT(espClient);

#if 0
#include <SPI.h>
#include <PN532_SPI.h>

PN532_SPI pn532spi(SPI, port);
NfcAdapter nfc = NfcAdapter(pn532spi);
#else 1
#define NDEF_DEBUG

#include <PN532_HSU.h>

//Rotines related with the configuration of the RFID reader PN532
HardwareSerial sSerial(port);
// If using ESP32 and Serial2 is not working follow the steps to use hardware serial to solve
the problem (https://esp32.com/viewtopic.php?t=4738)
PN532_HSU pn532hsu(sSerial);
NfcAdapter nfc(pn532hsu);
#else
#include <Wire.h>

```

```
#include <PN532_I2C.h>

PN532_I2C pn532_i2c(Wire);
NfcAdapter nfc = NfcAdapter(pn532_i2c);
#endif

Pn532NfcReader* reader = new Pn532NfcReader(&nfc);
```

APÊNDICE G TEMPLATE_READER_IPNRD.INO

```
String status = "", topic_status = "";
int sizearray[] = {1, 2};

/** Input of Incidence Matrix
 * The size of Incidence Matrix must contain the number of transitions and the number of
 * places in PetriNet
 */
int8_t* mIncidenceMatrix = new int8_t[2]{-1, 1};

/** Input of Token Vector
 * The size of Token Vector must contain the number of places in PetriNet
 */
uint16_t* mStartingTokenVector = new uint16_t[2]{1, 0};

uint32_t tagId = 0xFFFFFFFF;

void setup(void) {
  Serial.begin(115200);
  delay(1000);

  /** START THE CONNECTION TO WIFI
   * This call start the method connection that try to connect to SSID WiFi. If WiFi is not
   * connected try another SSID.
   * Commands:
   *   setWiFi(const char* ssid, const char* password)
   *   setWiFi(const char* ssid, wpa2_auth_method_t method, const char* identity, const
   char* username, const char* password)
   *   connect()
   *   status()
   */
  do {
    username == "XXXX" ? ewifi.setWiFi(SSID, password):ewifi.setWiFi(SSID,
WPA2_AUTH_PEAP, anonymous, username, password);
    ewifi.connect();

    if(ewifi.status() != WL_CONNECTED) printf("\nConnection failed (%s)...\n", SSID);
    else {break;}
  } while(true);

  printf("Connected to %s!\n", SSID);

  // Setup of Server and Subscribe
  MQTT.setServer(MQTTServer, MQTTPort);
  MQTT.setCallback(callback);
}

/* LOOP FUNCTION TO RUNNING PROGRAM */
```

```

void loop() {
  if (!MQTT.connected()) conectarMQTT();

  MQTT.loop();
}

/* CALLBACK FUNCTION FOR SUBSCRIBE TOPICS */
void callback(char* topic, byte* payload, unsigned int length) {
  String msg = "";
  String msg_topic = "";

  for(int i = 0; i < strlen(topic); i++) msg_topic += (char)topic[i];
  for(int i = 0; i < length; i++) msg += (char)payload[i];

  printf("Tópico: %s\tMensagem: %s\n", msg_topic.c_str(), msg.c_str());

  if (status != msg || topic_status != msg_topic) {
    status = msg;
    topic_status = msg_topic;
    msg.toLowerCase();
    if (!strcmp(msg_topic.c_str(), "PELMSVAR1")) {
      sizearray[0] = msg.toInt();
    } else if (!strcmp(msg_topic.c_str(), "PELMSVAR2")) {
      sizearray[1] = msg.toInt();
    } else if (!strcmp(msg_topic.c_str(), "PELMSVAR3")) {
      resize(mIncidenceMatrix, sizearray[0]*sizearray[1], msg);
    } else if (!strcmp(msg_topic.c_str(), "PELMSVAR4")) {
      resize(mStartingTokenVector, sizearray[0], msg);

      // printf("Places: %d\nTransition: %d\nReader: %s\nIncidence Matrix: {", sizearray[0],
      sizearray[1], readerID.c_str());
      // for (int i = 0; i < (sizearray[0]*sizearray[1]); i++)
      //   i == (sizearray[0]*sizearray[1] - 1) ? printf("%d}\nMarking Vector: {",
      mIncidenceMatrix[i]):printf("%d, ", mIncidenceMatrix[i]);
      // for (int i = 0; i < sizearray[0]; i++)
      //   i == (sizearray[0] - 1) ? printf("%d}\n", mStartingTokenVector[i]):printf("%d, ",
      mStartingTokenVector[i]);

    }
  }
}

void start_PNRD(){
  printf("Starting Tag Reader!!\n");

  Pnrd pnrd = Pnrd(reader,sizearray[0], sizearray[1], false, true);

  reader->initialize();

  //Setting of the application Petri net approach

```

```

pnrd.setTokenVector(mStartingTokenVector);
pnrd.setIncidenceMatrix(mIncidenceMatrix);

//Setting of the classic iPNRD approach
pnrd.setAsTagInformation(PetriNetInformation::FIRE_VECTOR);
pnrd.setAsTagInformation(PetriNetInformation::TAG_HISTORY);

FireError fireError;

//In case of a successful reading
while (pnrd.getData() != ReadError::NO_ERROR);

//Checks if it's a new tag
if (tagId != pnrd.getTagId()) {
    tagId = pnrd.getTagId();
    Serial.print("\nNew tag. Id code: ");
    Serial.println(tagId, HEX);

    //Realização do disparo contido na etiqueta
    fireError = pnrd.fire();
}

switch (fireError) {
    case FireError::NO_ERROR:
        Serial.println("Disparo bem sucedido.");

        //Mostrar o vetor de marcações resultante do disparo
        pnrd.printTokenVector();

        //Salvar a nova informação na etiqueta
        if (pnrd.saveData() == WriteError::NO_ERROR) {
            publish("NO_ERROR");
            Serial.println("Informação atualizada.");
        } else Serial.println("Erro na atualização da etiqueta.");

        break;
    case FireError::PRODUCE_EXCEPTION:
        publish("Exception");
        Serial.println("Erro: disparo gerou exceção.");

        break;
    case FireError::CONDITIONS_ARE_NOT_APPLIED:
        publish("Error: 001");
        Serial.println("Erro: condições não são satisfeitas.");

        break;
}
}

void publish(String error) {

```

```

char text[20];

snprintf(text, 20, "%s", tagId);
while (!MQTT.publish("PELMSVAR5", text))
    if (!MQTT.connected()) conectarMQTT();

while (!MQTT.publish("PELMSVAR6", pelmsReader.c_str()))
    if (!MQTT.connected()) conectarMQTT();

snprintf(text, 20, "%s", error);
while (!MQTT.publish("PELMSVAR7", text))
    if (!MQTT.connected()) conectarMQTT();

while (!MQTT.publish("PELMSVAR8", pelmsAntenna.c_str()))
    if (!MQTT.connected()) conectarMQTT();

while (!MQTT.publish("PELMSVAR9", pelmsPosition.c_str()))
    if (!MQTT.connected()) conectarMQTT();
}

/* MQTT FUNCTION FOR CONNECT TO SERVER */
void conectarMQTT() {
    // printf("%s\n", MQTTClientid);
    while (!MQTT.connected())
        if (MQTT.connect(MQTTClientid)) MQTT.subscribe(topic);
}

/* RESIZE THE MATRIX */
void resize(int8_t *&original, int setSize, String msg) {
    int8_t *temporiginal = new int8_t[setSize];
    int j = 0;
    volatile bool flag = false;

    for (int i = 0; i < msg.length(); i++) {
        String text = String(msg.charAt(i));
        if (strcmp(text.c_str(), "{") && strcmp(text.c_str(), ",") && strcmp(text.c_str(), " ") &&
            strcmp(text.c_str(), "}") && strcmp(text.c_str(), "-")) {
            temporiginal[j] = flag ? (-text.toInt()): (text.toInt());
            flag = false;
            j++;
        } else if (!strcmp(text.c_str(), "-")) {
            flag = true;
        }
    }

    delete[] original;
    original = temporiginal; //point old array to new array
}

void resize(uint16_t *&original, int setSize, String msg) {

```

```
uint16_t *temporiginal = new uint16_t[setSize];
int j = 0;

for (int i = 0; i < msg.length(); i++) {
    String text = String(msg.charAt(i));
    if (strcmp(text.c_str(), "{") && strcmp(text.c_str(), ",") && strcmp(text.c_str(), " ") &&
    strcmp(text.c_str(), "}")) {
        temporiginal[j] = text.toInt();
        j++;
    }
}

delete[] original;
original = temporiginal; //point old array to new array
}
```

APÊNDICE H TEMPLATE_READER_PNRD.INO

```

String status = "", topic_status = "";
volatile bool flag = false;
int sizearray[] = {1, 2};

uint32_t tagId = 0xFFFFFFFF;

void setup(void) {
  Serial.begin(115200);
  delay(1000);

  /** START THE CONNECTION TO WIFI
   * This call start the method connection that try to connect to SSID WiFi. If WiFi is not
   connected try another SSID.
   * Commands:
   *   setWiFi(const char* ssid, const char* password)
   *   setWiFi(const char* ssid, wpa2_auth_method_t method, const char* identity, const
char* username, const char* password)
   *   connect()
   *   status()
   */
  do {
    username == "XXXX" ? ewifi.setWiFi(SSID, password):ewifi.setWiFi(SSID,
WPA2_AUTH_PEAP, anonymous, username, password);
    ewifi.connect();

    if(ewifi.status() != WL_CONNECTED) printf("\nConnection failed (%s)...\n", SSID);
    else {break;}
  } while(true);

  printf("Connected to %s!\n", SSID);

  // Setup of Server and Subscribe
  MQTT.setServer(MQTTServer, MQTTPort);
  MQTT.setCallback(callback);
}

/* LOOP FUNCTION TO RUNNING PROGRAM */
void loop() {
  if (!MQTT.connected()) conectarMQTT();

  MQTT.loop();
}

/* CALLBACK FUNCTION FOR SUBSCRIBE TOPICS */
void callback(char* topic, byte* payload, unsigned int length) {
  String msg = "";
  String msg_topic = "";

```

```

for(int i = 0; i < strlen(topic); i++) msg_topic += (char)topic[i];
for(int i = 0; i < length; i++) msg += (char)payload[i];

printf("Tópico: %s\tMensagem: %s\n", msg_topic.c_str(), msg.c_str());

if (status != msg || topic_status != msg_topic) {
    status = msg;
    topic_status = msg_topic;
    msg.toLowerCase();
    if (!strcmp(msg_topic.c_str(), "PELMSVAR1")) {
        sizearray[0] = msg.toInt();
    } else if (!strcmp(msg_topic.c_str(), "PELMSVAR2")) {
        sizearray[1] = msg.toInt();

        // printf("Places: %d\nTransition: %d\nReader: %s\nAntenna: %s\nFire Position: %d\n",
sizearray[0], sizearray[1], pelmsReader.c_str(), pelmsAntenna.c_str(),
pelmsPosition.c_str());

        start_PNRD();
    }
}

void start_PNRD(){
    printf("Starting Tag Reader!!\n");

    Pnrd pnrd = Pnrd(reader, sizearray[0], sizearray[1]);

    reader->initialize();

    //Configuração para a abordagem PNRD clássica
    pnrd.setAsTagInformation(PetriNetInformation::TOKEN_VECTOR);
    pnrd.setAsTagInformation(PetriNetInformation::ADJACENCY_LIST);

    FireError fireError;

    //In case of a successful reading
    while (pnrd.getData() != ReadError::NO_ERROR);

    //Checks if it's a new tag
    if (tagId != pnrd.getTagId()) {
        tagId = pnrd.getTagId();
        Serial.print("\nNew tag. Id code: ");
        Serial.println(tagId, HEX);

        //Realização do disparo contido na etiqueta
        fireError = pnrd.fire(pelmsPosition.toInt());
    }
}

```

```

switch (fireError) {
case FireError::NO_ERROR:
    Serial.println("Disparo bem sucedido.");

    //Mostrar o vetor de marcações resultante do disparo
    pnr.printTokenVector();

    //Salvar a nova informação na etiqueta
    if (pnrd.saveData() == WriteError::NO_ERROR) {
        publish("NO_ERROR");
        Serial.println("Informação atualizada.");
    } else Serial.println("Erro na atualização da etiqueta.");

    break;
case FireError::PRODUCE_EXCEPTION:
    publish("Exception");
    Serial.println("Erro: disparo gerou exceção.");

    break;
case FireError::CONDITIONS_ARE_NOT_APPLIED:
    publish("Error: 001");
    Serial.println("Erro: condições não são satisfeitas.");

    break;
}
}

void publish(String error) {
    char text[20];

    snprintf(text, 20, "%s", tagId);
    while (!MQTT.publish("PELMSVAR3", text))
        if (!MQTT.connected()) conectarMQTT();

    while (!MQTT.publish("PELMSVAR4", pelmsReader.c_str()))
        if (!MQTT.connected()) conectarMQTT();

    snprintf(text, 20, "%s", error);
    while (!MQTT.publish("PELMSVAR5", text))
        if (!MQTT.connected()) conectarMQTT();

    while (!MQTT.publish("PELMSVAR6", pelmsAntenna.c_str()))
        if (!MQTT.connected()) conectarMQTT();

    while (!MQTT.publish("PELMSVAR7", pelmsPosition.c_str()))
        if (!MQTT.connected()) conectarMQTT();
}

/* MQTT FUNCTION FOR CONNECT TO SERVER */
void conectarMQTT() {

```

```
// printf("%s\n", MQTTClientid);  
while (!MQTT.connected())  
    if (MQTT.connect(MQTTClientid)) MQTT.subscribe(topic);  
}
```

APÊNDICE I TEMPLATE_RECORDER_IPNRD.INO

```
String status = "", topic_status = "";
int sizearray[] = {1, 2};

/** Input of Fire Vector:
 * The Fire Vector must contain the number of transitions in the PetriNet
 */
uint16_t* mFireVector = new uint16_t[1]{1};

void setup(void) {
  Serial.begin(115200);
  delay(1000);

  /** START THE CONNECTION TO WIFI
   * This call start the method connection that try to connect to SSID WiFi. If WiFi is not
   connected try another SSID.
   * Commands:
   *   setWiFi(const char* ssid, const char* password)
   *   setWiFi(const char* ssid, wpa2_auth_method_t method, const char* identity, const
   char* username, const char* password)
   *   connect()
   *   status()
   */
  do {
    username == "XXXX" ? ewifi.setWiFi(SSID, password):ewifi.setWiFi(SSID,
    WPA2_AUTH_PEAP, anonymous, username, password);
    ewifi.connect();

    if(ewifi.status() != WL_CONNECTED) printf("\nConnection failed (%s)...\n", SSID);
    else break;
  } while(true);

  printf("Connected to %s!\n", SSID);

  // Setup of Server and Subscribe
  MQTT.setServer(MQTTServer, MQTTPort);
  MQTT.setCallback(callback);
}

/* LOOP FUNCTION TO RUNNING PROGRAM */
void loop() {
  if (!MQTT.connected()) conectarMQTT();

  MQTT.loop();
}

/* CALLBACK FUNCTION FOR SUBSCRIBE TOPICS */
void callback(char* topic, byte* payload, unsigned int length) {
```

```

String msg = "";
String msg_topic = "";

for(int i = 0; i < strlen(topic); i++) msg_topic += (char)topic[i];
for(int i = 0; i < length; i++) msg += (char)payload[i];

printf("Tópico: %s\tMensagem: %s\n", msg_topic.c_str(), msg.c_str());

if (status != msg || topic_status != msg_topic) {
    status = msg;
    topic_status = msg_topic;
    msg.toLowerCase();
    if (!strcmp(msg_topic.c_str(), "PELMSVAR1")) {
        sizearray[0] = msg.toInt();
    } else if (!strcmp(msg_topic.c_str(), "PELMSVAR2")) {
        sizearray[1] = msg.toInt();
    } else if (!strcmp(msg_topic.c_str(), "PELMSVAR3")) {
        resize(mFireVector, sizearray[1], msg);

        // printf("Places: %d\nTransition: %d\nFire Position: {", sizearray[0], sizearray[1]);
        // for (int i = 0; i < sizearray[1]; i++)
        //     i == (sizearray[1] - 1) ? printf("%d}\n", mFireVector[i]):printf("%d, ",
mFireVector[i]);
        printf("Starting Tag Recording!!\n");

        // Preparing the tag information
        eraseRFID();
        cleanRFID();
        formatRFID();
        recordRFID();
    }
}

void eraseRFID() {
    #if 0
        PN532 tag(pn532spi);
    #elif 1
        PN532 tag(pn532hsu);
    #else
        PN532 tag(pn532_i2c);
    #endif

    nfc.begin();
    tag.begin();

    // Set the max number of retry attempts to read from a card
    // This prevents us from waiting forever for a card, which is
    // the default behaviour of the PN532.
    tag.setPassiveActivationRetries(0xFF);

```

```

// configure board to read RFID tags
tag.SAMConfig();

uint8_t uid[] = {0, 0, 0, 0, 0, 0, 0};
uint8_t uidLength;
String tagID = "";

printf("Please place the Tag!!!\n");

// Identifier if tag is present
while (!nfc.tagPresent()) delay(500);

tag.readPassiveTargetID(PN532_MIFARE_ISO14443A, &uid[0], &uidLength);
for(int i = 0; i < uidLength; i++) tagID += String(uid[i], HEX);
tagID.toUpperCase();
printf("Tag Presented (TagID: %s)!!!\n", tagID);

// Erase the tag returning the success of erasing
printf("Erasing the Tag.\n");
bool success = nfc.erase();

// Return the message of erasing
printf(success ? "Tag erased successfully.\n":"Error to erasing the tag!\n");
}

void cleanRFID() {
  nfc.begin();

  // Identifier if tag is present
  while (!nfc.tagPresent()) delay(500);

  // Return the tag do Factory and returning the success of Factory Setup
  printf("Restoring the tag to factory setup.\n");

  bool success = nfc.clean();

  printf(success ? "Successfully restoring the tag to factory setup.\n":"Tag is already in factory
  setup!\n");
}

void formatRFID() {
  nfc.begin();

  // Identifier if tag is present
  while (!nfc.tagPresent()) delay(500);

  // Format the tag returning the success of formatting
  printf("Formating the Tag to NDEF format.\n");
}

```

```

bool success = nfc.format();

printf(success ? "Tag successfully formatted as NDEF format.\n":"Tag is already formatted
as NDEF format!\n");
}

/* Not Used */
void readRFID() {
    // Read the information of the Tag
    NfcTag tag = nfc.read();

    // Print the information of the Tag
    tag.print();
}

void recordRFID() {
    /** Creation of the reader and PNRD objects
     *   Pnrd(readerPointer, num_places, num_transitions, num_max_of_inputs,
num_max_of_outputs, hasConditions, hasTagHistory)
     */
    Pnrd pnr = Pnrd(reader, sizearray[0], sizearray[1]); //leitor, no estados e no transicoes

    // Initializing PN532reader
    reader->initialize();

    //Defining the Fire Vector to be recorded
    pnr.setFireVector(mFireVector);

    //Setting of the classic iPNRD approach
    pnr.setAsTagInformation(PetriNetInformation::FIRE_VECTOR);
    pnr.setAsTagInformation(PetriNetInformation::TAG_HISTORY);
    Serial.print("\nInitial recording of iPNRD tags.\n");

    // Identifier if tag is present
    while (!nfc.tagPresent()) delay(500);

    printf("Storing the iPNRD in Tag!\n");
    // Cormfirming if data is saved
    if(pnr.saveData() == WriteError::NO_ERROR) printf("iPNRD is stored in the tag!");
    else printf("Error to storing iPNRD in the tag!");
}

/* MQTT FUNCTION FOR CONECT TO SERVER */
void conectarMQTT() {
    // printf("%s\n", MQTTClientid);
    while (!MQTT.connected())
        if (MQTT.connect(MQTTClientid)) MQTT.subscribe(topic);
}

/* RESIZE THE MATRIX */

```

```
void resize(uint16_t *&original, int setSize, String msg) {
    uint16_t *temporiginal = new uint16_t[setSize];
    int j = 0;

    for (int i = 0; i < msg.length(); i++) {
        String text = String(msg.charAt(i));
        if (strcmp(text.c_str(), "{") && strcmp(text.c_str(), ",") && strcmp(text.c_str(), " ") &&
            strcmp(text.c_str(), "}")) {
            temporiginal[j] = text.toInt();
            j++;
        }
    }

    delete[] original;
    original = temporiginal; //point old array to new array
}
```

APÊNDICE J TEMPLATE_RECORDER_PNRD.INO

```
String status = "", topic_status = "";
int sizearray[] = {1, 2};

/** Input of Incidence Matrix
 * The size of Incidence Matrix must contain the number of transitions and the number of
 * places in PetriNet
 */
int8_t* mIncidenceMatrix = new int8_t[2]{-1,1};

/** Input of Token Vector
 * The size of Token Vector must contain the number of places in PetriNet
 */
uint16_t* mStartingTokenVector = new uint16_t[2]{1,0};

void setup(void) {
  Serial.begin(115200);
  delay(1000);

  /** START THE CONNECTION TO WIFI
   * This call start the method connection that try to connect to SSID WiFi. If WiFi is not
   * connected try another SSID.
   * Commands:
   *   setWiFi(const char* ssid, const char* password)
   *   setWiFi(const char* ssid, wpa2_auth_method_t method, const char* identity, const
   * char* username, const char* password)
   *   connect()
   *   status()
   */
  do {
    username == "XXXX" ? ewifi.setWiFi(SSID, password):ewifi.setWiFi(SSID,
WPA2_AUTH_PEAP, anonymous, username, password);
    ewifi.connect();

    if(ewifi.status() != WL_CONNECTED) printf("\nConnection failed (%s)...\n", SSID);
    else break;
  } while(true);

  printf("Connected to %s!\n", SSID);

  // Setup of Server and Subscribe
  MQTT.setServer(MQTTServer, MQTTPort);
  MQTT.setCallback(callback);
}

/* LOOP FUNCTION TO RUNNING PROGRAM */
void loop() {
  if (!MQTT.connected()) conectarMQTT();
}
```

```

MQTT.loop();
}

/* CALLBACK FUNCTION FOR SUBSCRIBE TOPICS */
void callback(char* topic, byte* payload, unsigned int length) {
    String msg = "";
    String msg_topic = "";

    for(int i = 0; i < strlen(topic); i++) msg_topic += (char)topic[i];
    for(int i = 0; i < length; i++) msg += (char)payload[i];

    printf("Tópico: %s\tMensagem: %s\n", msg_topic.c_str(), msg.c_str());

    if (status != msg || topic_status != msg_topic) {
        status = msg;
        topic_status = msg_topic;
        msg.toLowerCase();
        if (!strcmp(msg_topic.c_str(), "PELMSVAR1")) {
            sizearray[0] = msg.toInt();
        } else if (!strcmp(msg_topic.c_str(), "PELMSVAR2")) {
            sizearray[1] = msg.toInt();
        } else if (!strcmp(msg_topic.c_str(), "PELMSVAR3")) {
            resize(mStartingTokenVector, sizearray[0], msg);
        } else if (!strcmp(msg_topic.c_str(), "PELMSVAR4")) {
            resize(mIncidenceMatrix, sizearray[0]*sizearray[1], msg);

            // printf("Places: %d\nTransition: %d\nIncidence Matrix: {", sizearray[0], sizearray[1]);
            // for (int i = 0; i < (sizearray[0]*sizearray[1]); i++)
            //     i == (sizearray[0]*sizearray[1] - 1) ? printf("%d}\nMarking Vector: {",
            mIncidenceMatrix[i]):printf("%d, ", mIncidenceMatrix[i]);
            // for (int i = 0; i < sizearray[0]; i++)
            //     i == (sizearray[0] - 1) ? printf("%d}\n", mStartingTokenVector[i]):printf("%d, ",
            mStartingTokenVector[i]);
            printf("Starting Tag Recording!!\n");

            // Preparing the tag information
            eraseRFID();
            cleanRFID();
            formatRFID();
            recordRFID();
        }
    }
}

void eraseRFID() {
    #if 0
        PN532 tag(pn532spi);
    #elif 1
        PN532 tag(pn532hsu);
    #endif
}

```

```

#else
    PN532 tag(pn532_i2c);
#endif

nfc.begin();
tag.begin();

// Set the max number of retry attempts to read from a card
// This prevents us from waiting forever for a card, which is
// the default behaviour of the PN532.
tag.setPassiveActivationRetries(0xFF);

// configure board to read RFID tags
tag.SAMConfig();

uint8_t uid[] = {0, 0, 0, 0, 0, 0, 0};
uint8_t uidLength;
String tagID = "";

printf("Please place the Tag!!!\n");

// Identifier if tag is present
while (!nfc.tagPresent()) delay(500);

tag.readPassiveTargetID(PN532_MIFARE_ISO14443A, &uid[0], &uidLength);
for(int i = 0; i < uidLength; i++) tagID += String(uid[i], HEX);
tagID.toUpperCase();
printf("Tag Presented (TagID: %s)!!!\n", tagID);

// Erase the tag returning the success of erasing
printf("Erasing the Tag.\n");
bool success = nfc.erase();

// Return the message of erasing
printf(success ? "Tag erased successfully.\n":"Error to erasing the tag!\n");
}

void cleanRFID() {
    nfc.begin();

    // Identifier if tag is present
    while (!nfc.tagPresent()) delay(500);

    // Return the tag do Factory and returning the success of Factory Setup
    printf("Restoring the tag to factory setup.\n");

    bool success = nfc.clean();

    printf(success ? "Successfully restoring the tag to factory setup.\n":"Tag is already in factory
    setup!\n");
}

```

```

}

void formatRFID() {
    nfc.begin();

    // Identifier if tag is present
    while (!nfc.tagPresent()) delay(500);

    // Format the tag returning the success of formatting
    printf("Formating the Tag to NDEF format.\n");

    bool success = nfc.format();

    printf(success ? "Tag successfully formated as NDEF format.\n":"Tag is already formated
as NDEF format!\n");
}

/* Not Used */
void readRFID() {
    // Read the information of the Tag
    NfcTag tag = nfc.read();

    // Print the information of the Tag
    tag.print();
}

void recordRFID() {
    /** Creation of the reader and PNRD objects
    *   Pnrd(readerPointer,    num_places,    num_transitions,    num_max_of_inputs,
num_max_of_outputs, hasConditions, hasTagHistory)
    */
    Pnrd pnrld = Pnrd(reader, sizeof array[0], sizeof array[1]); //leitor, no estados e no transicoes

    // Initializing PN532reader
    reader->initialize();

    //Defining the Incidence Matrix and the Token Vector to be recorded
    pnrld.setIncidenceMatrix(mIncidenceMatrix);
    pnrld.setTokenVector(mStartingTokenVector);

    //Setting of the classic PNRD approach
    pnrld.setAsTagInformation(PetriNetInformation::TOKEN_VECTOR);
    pnrld.setAsTagInformation(PetriNetInformation::ADJACENCY_LIST);
    Serial.print("\nInitial recording of PNRD tags.\n");

    // Identifier if tag is present
    while (!nfc.tagPresent()) delay(500);

    printf("Storing the PNRD in Tag!\n");
    // Cormfirming if data is saved

```

```

if(pnrd.saveData() == WriteError::NO_ERROR) printf("PNRD is stored in the tag!");
else printf("Error to storing PNRD in the tag!");
}

/* MQTT FUNCTION FOR CONECT TO SERVER */
void conectarMQTT() {
    // printf("%s\n", MQTTClientid);
    while (!MQTT.connected())
        if (MQTT.connect(MQTTClientid)) MQTT.subscribe(topic);
}

/* RESIZE THE MATRIX */
void resize(int8_t *&original, int setSize, String msg) {
    int8_t *temporiginal = new int8_t[setSize];
    int j = 0;
    volatile bool flag = false;

    for (int i = 0; i < msg.length(); i++) {
        String text = String(msg.charAt(i));
        if (strcmp(text.c_str(), "{") && strcmp(text.c_str(), ",") && strcmp(text.c_str(), " ") &&
            strcmp(text.c_str(), "}") && strcmp(text.c_str(), "-")) {
            temporiginal[j] = flag ? (-text.toInt()): (text.toInt());
            flag = false;
            j++;
        } else if (!strcmp(text.c_str(), "-")) {
            flag = true;
        }
    }

    delete[] original;
    original = temporiginal; //point old array to new array
}

void resize(uint16_t *&original, int setSize, String msg) {
    uint16_t *temporiginal = new uint16_t[setSize];
    int j = 0;

    for (int i = 0; i < msg.length(); i++) {
        String text = String(msg.charAt(i));
        if (strcmp(text.c_str(), "{") && strcmp(text.c_str(), ",") && strcmp(text.c_str(), " ") &&
            strcmp(text.c_str(), "}") && strcmp(text.c_str(), "-")) {
            temporiginal[j] = text.toInt();
            j++;
        }
    }

    delete[] original;
    original = temporiginal; //point old array to new array
}

```

APÊNDICE K TEMPLATE_WIFIPASSWORD.H

```

/**
 * Created on Dec, 2024
 * @headerfile: PNRD recorder
 *
 * @details: This code receive message from MQTT protocol and set the variables of
 * the PNRD/iPNRD to be stored in a tag.
 * 1- MQTT set the variables of PNRD/iPNRD
 * 2- When tag presented it erase, restore to factory and format the tag as NDEF format
 * 3- Then it stores the PNRD/iPNRD in the tag using NDEF format
 *
 * @author: Álisson Carvalho Vasconcelos
 * @authors updater:
 * @authors github: AlissonCV
 *
 * @copyright: Free for All
 */

/** DEFINITION OF WiFi DATA
 * Definition of macros SSID, username and password
 */
#define SSID "PELMSVAR01"
#define username "PELMSVAR02"
#define password "PELMSVAR03"
#define anonymous "PELMSVAR04"

/** DEFINITION OF MQTT DATA
 * Definition of macros MQTTClientid, MQTTServer and MQTTPort
 */
#define MQTTClientid "PELMSVAR05"
#define MQTTServer "PELMSVAR06"
#define MQTTPort PELMSVAR07

/* DEFINITION OF READER PORT INPUT */
#define port PELMSVAR08

/* DECLARATION OF TOPICS VARIABLES */
const char* topic = "PELMSVAR09";

```