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

DANIEL BARBOSA PEREIRA

SISTEMA DE GERENCIAMENTO DA BIBLIOTECA ARDUINO PARA PNRD POR COMPUTAÇÃO EM NUVEM

UBERLÂNDIA
2022

DANIEL BARBOSA PEREIRA

SISTEMA DE GERENCIAMENTO DA BIBLIOTECA ARDUINO PARA PNRD POR COMPUTAÇÃO EM NUVEM

Trabalho de Conclusão de Curso apresentado no curso de graduação em Engenharia Mecatrônica da Universidade Federal de Uberlândia, como parte dos requisitos para a obtenção de título de **BACHAREL EM ENGENHARIA MECATRÔNICA**

Área de concentração: Engenharia Mecatrônica

Orientador: Prof. Dr. José Jean-Paul Zanlucchi de Souza Tavares

UBERLÂNDIA
2022

Este trabalho é dedicado a meus pais, amigos e a todos que foram essenciais na minha formação e crescimento.

AGRADECIMENTOS

À Deus que permitiu e proporcionou que eu chegasse aqui.

Aos meus pais que sempre me apoiaram e sempre se mantiveram presentes. "Vai dar certo", foi a frase que mais escutei na minha graduação.

Às escolhas da vida, que direta ou indiretamente me fizeram ser quem sou e chegar onde cheguei.

Ao professor Dr. José Jean-Paul Zanlucchi de Souza Tavares, que me orientou e me inpirou a trabalhar nesse projeto. A criação dessa nova tecnologia se deve muito e esse profissional.

Aos meus colegas e amigos que, durante a graduação e me ajudaram a atravessar este período da melhor forma possível.

Aos professores que foram essenciais para minha formação.

"Nós vamos vencer. Não há outra opção"
Luide Matos

RESUMO

Em um mundo pós-pandêmico, sistemas onde não há uma propagação de doenças se tornam cada vez mais necessários. Nesse caso, tecnologias como computação em Nuvem e RFID são alternativas para processos industriais. Nessa direção, as redes de Petri inseridas em RFID, as PNRD, podem ser úteis nesses cenários. Esse trabalho apresenta o projeto e a implementação de uma nova versão do software PALMS para que o mesmo possa trabalhar em Cloud ou Nuvem. Utilizando código aberto, um microcontrolador de baixo custo e sistemas de leitura e escrita RFID, um software que integra essas tecnologias foi implementado. O resultado foi a criação de um sistema descentralizado, escalável, robusto e autônomo.

Palavras-chave: Rede de Petri, Computação por Nuvem, Arduino®, MQTT

ABSTRACT

In a post-pandemic world, systems where there is no spread of disease become increasingly necessary. In this case, technologies such as Cloud computing and RFID are alternatives for industrial processes. In this direction, the Petri nets inserted in RFID, the PNRD can be useful in these scenarios. This work presents the design and implementation of a new version of the PALMS software so that it can work in Cloud. Using open source, a low cost microcontroller and RFID reading and writing systems, a software that integrates these technologies was implemented. The result was the creation of a decentralized, scalable, robust and autonomous system.

Key-words:Petri Net, Cloud Computing, Arduino®, MQTT.

LISTA DE ILUSTRAÇÕES

Figura 1 – Tag RFID em formato de chaveiro usada no projeto 14
Figura 2 – Tag RFID em formato de cartão usada no projeto 14
Figura 3 – Esquema de funcionamento tag RFID, (PAULA, 2021) 15
Figura 4 – Representação de um semáforo em redes de petri no estado verde, (PAULA, 2021) 15
Figura 5 – Disparo da transição T_0 movimentando o token de P_0 para P_1, (PAULA, 2021) 16
Figura 6 – Representação simples de uma Rede de Lugar/Transição (KINDLER, 2006) 18
Figura 7 – Código PNML relativo a figura 6 (KINDLER, 2006) 19
Figura 8 – Associação dos termos da Equação (2.1) com os elementos onde serão armazenados nas abordagens PNRD e iPNRD, (FONSECA et al., 2018) 20
Figura 9 – Esquema de funcionamento de um *Broker* MQTT, (MQTT, 2022), traduzido pelo autor 21
Figura 10 – Esquema de funcionamento do PALMS 23
Figura 11 – Shield Ethernet com cartão SD 24
Figura 12 – Criação do *Broker* 24
Figura 13 – Planos para criação do *Broker* 24
Figura 14 – Dados para criação do *Broker* 25
Figura 15 – *Broker* MQTT utilizado no projeto 26
Figura 16 – Arduino® UNO escolhido para comunicação Arduino®-Python 27
Figura 17 – Arduino® Mega com Shield Ethernet utilizado para comunicação Arduino®-Python 28
Figura 18 – Leitor PN532 RFID utilizado 29
Figura 19 – Montagem realizada para inicialização da Tag 30
Figura 20 – Montagem realizada para leitura da Tag 31
Figura 21 – Carregando a PNRD no PALMS 32
Figura 22 – Modo SETUP do PALMS na criação do arquivo setup 33
Figura 23 – Modo RUNTIME do PALMS 34
Figura 24 – Arduino® Iniciador de Tags salvando os dados e lendo o cartão SD . . 34
Figura 25 – Tag corretamente configurada com os dados 35
Figura 26 – Lendo a Tag RDIF 35
Figura 27 – PALMS recebendo informação NO ERROR da Tag 36
Figura 28 – PALMS recebendo informação CONDITIONS ARE NOT APLIED da Tag 36

LISTA DE ABREVIATURAS E SIGLAS

FTP	*File Transfer Protocol*
IoT	Internet das coisas
IP	Protocolo de Internet
iPNRD	PNRD invertida
MAPL	Laboratório de Planejamento Automático de Manufatura
MQTT	*Message Queuing Telemetry Transport*
PALMS	Biblioteca de Arduino para gerenciamento de PNRD/iPNRD
PNML	Ferramenta de Modelagem para Redes de Petri
PNRD	Redes de Petri inseridas em RFID
RFID	Identificação por Rádio-Frequência
RP	Rede de Petri
SD	*Secure Digital*
XML	Formato de Arquivo Universal com Dados Organizados

LISTA DE SÍMBOLOS

P	Número finito de estados
T	Conjunto de transições
A	Conjunto de arcos
W	Função peso
M_k	Vetor de marcações
A^T	Matriz de adjacência
u_k	Vetor de disparos
T_k	Transição k, onde k = 0, 1, 2, . . . , n

SUMÁRIO

1 INTRODUÇÃO 12
1.1 OBJETIVOS DO PROJETO 13
1.1.1 Objetivos Específicos - Requisitos de Projeto 13
1.2 DOS TRABALHOS ANTERIORES 13
1.3 ESTRUTURA DO TRABALHO 13
2 FUNDAMENTAÇÃO TEÓRICA 14
2.1 INTRODUÇÃO 14
2.2 TECNOLOGIA RFID 14
2.3 REDES DE PETRI 15
2.3.1 Representação Gráfica 16
2.3.2 Conceito de Disparo 16
2.3.3 Representação Matemática 16
2.3.4 PNML 18
2.4 PNRD 18
2.4.1 Exceções 18
2.4.2 PNRD INVERTIDA (iPNRD) 20
2.5 MQTT 20
3 METODOLOGIA 22
3.1 INTRODUÇÃO 22
3.1.1 PALMS 22
3.2 ESTUDOS INICIAIS 22
3.2.1 Criando o *Broker* 22
3.3 COMUNICAÇÃO PUBLICADOR-SUBSCRITOR 25
3.4 COMUNICAÇÃO PYTHON-ARDUINO® 26
3.5 IMPLEMENTAÇÃO DO PALMS-MQTT 26
3.6 IMPLEMENTAÇÃO DO ARDUINO® INICIADOR DE TAGS 27
3.7 IMPLEMENTAÇÃO DO ARDUINO® LEITOR 28
4 RESULTADOS 32
4.1 MODO SETUP 32
4.2 MODO RUNTIME 33
5 CONCLUSÃO 37
6 TRABALHOS FUTUROS 38
6.1 IMPLEMENTAÇÃO DE UMA VERSÃO QUE UTILIZA TANTO FTP QUANTO MQTT 38
6.2 IMPLEMENTAÇÃO DO PALMS-MQTT EM UMA *BLOCK-CHAIN* 38

6.3 IMPLEMENTAR UMA VERSÃO QUE PERMITA ALTERAÇÕES AO LONGO DA EXECUÇÃO 38
6.4 HOMOLOGAÇÃO E CERTIFICAÇÕES 38

REFERÊNCIAS 40
A – APÊNDICE - PROGRAMA PUBLICADOR EM PYTHON 41
B – APÊNDICE - PROGRAMA SUBSCRITOR EM PYTHON 44
C – APÊNDICE - PROGRAMA DO ARDUINO® INICIADOR DE TAGS 47
D – APÊNDICE - PROGRAMA DO ARDUINO® LEITOR . 55

1 INTRODUÇÃO

Em um mundo pós-pandêmico, sistemas onde não há uma propagação de doenças se tornam cada vez mais necessários. Um sistema onde não há contato entre as partes ou sistemas autônomos são a base das tecnologias que serão desenvolvidas nos próximos anos. Sistemas industriais com comunicações por rádio frequências e sistemas controlados por Nuvem fazem parte dessa nova forma de se produzir.

Vê-se na quarta revolução industrial diversos conceitos que se adequam as exigências dessa nova forma de produção. Conceitos como Internet das Coisas (IoT), computação em Nuvem, sistemas autônomos e *Big Data* tem grande papel na economia pós-pandemia.

Um dos conceitos é de IoT, que nos traz conectividade para qualquer coisa. Desde eletrodomésticos até robôs industriais de grande escala estão cada vez mais integrados. Se trata de um mundo onde objetos físicos, dados virtuais, meio ambiente e os seres humanos podem estar conectados e podem interagir um com o outro de qualquer lugar e a qualquer momento.(SUNDMAEKER et al., 2010).

As cadeias produtivas conectadas em Nuvem são a forma da quarta revolução industrial tornar os sistemas cada vez mais autônomos e distribuídos. A computação em Nuvem descreve um ambiente baseado na gigantesca rede de servidores, sendo estes, virtuais ou físicos.(TAURION, 2009). Com o avanço das tecnologias, os modelos produtivos com o tempo passarão a ser customizáveis e sob demanda, onde um servidor em Nuvem, dotado de uma Inteligência Artificial, capaz de analisar dados, irá solicitar um bem que seja mais adequado ao consumidor e em qualquer lugar do mundo.

Com a necessidade de cada vez menos contato, as tecnologias RFID se veem cada vez mais presentes. Desde bilhetes de ônibus, pagamentos por aproximação, detectores de presença, dentre outros meios, essa tecnologia está cada vez mais presente no mundo em que vivemos. Com esse grande avanço, podem surgir falhas, nesse caso um sistema onde o controle desses dados pode ser mais seguro, se vê cada vez mais necessário. A PNRD e a iPNRD são ótimas ferramentas de análise de processos, levando o controle dos mesmos para as fronteiras, onde falhas podem ser identificadas imediatamente por leitores RFID.(PIRES, 2019)

Esse trabalho visa unir todas essas tecnologias da quarta revolução industrial supracitadas de forma simples e barata, através de um código aberto de um programa e um microcontrolador. E como estruturar e gerenciar os dados na PNRD nos objetos físicos em processos dinâmicos? E como esse gerenciamento pode atender os paradigmas da quarta revolução industrial?

1.1 OBJETIVOS DO PROJETO

O projeto tem como objetivo Desenvolver nova versão (1.0) do software PALMS *(PNRD/iPNRD Arduino Library Management System)* para que o mesmo possa trabalhar em *Cloud* ou Nuvem.

1.1.1 Objetivos Específicos - Requisitos de Projeto

- Especificar e implementar o sistema PALMS (1.0);
- Desenvolver um estudo de caso utilizando o protocolo MQTT (Internet das Coisas) para troca de mensagens no sistema PALMS (1.0).

1.2 DOS TRABALHOS ANTERIORES

Esse trabalho usa a biblioteca para aplicações de PNRD e PNRD invertida embarcadas em Arduino criada por Carlos Eduardo Alves da Silva e membros do MAPL. (SILVA et al., 2017).

Esse trabalho é a continuação do trabalho de conclusão de curso desenvolvido por Roger Henrique Carrijo de Paula (PAULA, 2021), atualizando o sistema a IoT através de conexões com o protocolo MQTT.

1.3 ESTRUTURA DO TRABALHO

O restante do trabalho é dividido da seguinte maneira.

Fundamentação Teórica: contém as bases teóricas do trabalho, com detalhamento das tecnologias e conceitos utilizados.

Metodologia: descreve como a teoria descrita no capítulo anterior foi aplicada para a implementação do projeto e justifica as decisões tomadas ao longo do seu desenvolvimento.

Resultados: mostra os resultados obtidos com a implementação do projeto e a utilização do sistema.

Conclusão: analisa o impacto do trabalho no contexto de aplicação e como os objetivos do projeto foram cumpridos.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 INTRODUÇÃO

A solução desenvolvida nesse trabalho tem um aspecto amplo e interdisciplinar. Este capítulo apresenta a fundamentação teórica para o embasamento técnico do projeto.

2.2 TECNOLOGIA RFID

A tecnologia de identificação por radiofrequência, ou RFID, utiliza ondas eletromagnéticas para armazenar e ler informações de pequenos dispositivos chamados etiquetas ou tags. A figura 1 representa uma tag em formato de chaveiro e a figura 2 representa uma tag em formato de cartão.

Figura 1 – Tag RFID em formato de chaveiro usada no projeto

Figura 2 – Tag RFID em formato de cartão usada no projeto

Nesse sistema o leitor, acoplado a uma antena, envia um sinal eletromagnético buscando a tag. Quando encontrada, a tag recebe esse sinal, que é processado por um microchip, e transmitido de volta ao leitor. No leitor o sinal é demodulado e transformado em dados. Na figura 3 vemos um exemplo de funcionamento da tag RFID.

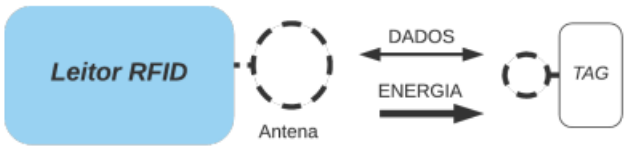

Figura 3 – Esquema de funcionamento tag RFID, (PAULA, 2021)

A utilização da tecnologia RFID é vantajosa para a identificação de objetos. As tags RFID tem fácil leitura, visto que a tag só precisa estar no alcance da antena, não sendo necessário contato entra as partes do sistema. Outra vantagem é o fato de que podem ser lidas várias tags de uma única vez, deixando o processo mais rápido.

Com fácil leitura e capacidade de armazenamento de dados, as tags RFID permitem que dados possam ser enviados e recebidos através das antenas, facilitando a comunicação entre sistemas.

2.3 REDES DE PETRI

As redes de Petri (RP) são um modelo conceitual gráfico e matemático introduzido por Carl Adam Petri em 1962 (MURATA, 1989). As redes de Petri podem ser utilizadas para modelar os mais diversos sistemas, sendo compostos por eventos assíncronos, concorrentes, paralelos, distribuídos ou não-determinísticos.

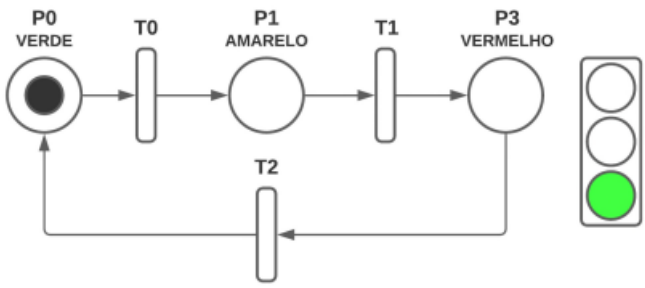

Figura 4 – Representação de um semáforo em redes de petri no estado verde, (PAULA, 2021)

2.3.1 Representação Gráfica

A representação gráfica das redes de Petri é uma forma de visualizar os processos através de elementos visuais.

A RP possui quatro elementos, os Lugares(P), Transições(T), Arcos(A) e fichas (tokens). Os lugares são tipicamente representados por circunferências, as transições por traços ou retângulos, os arcos por setas que ligam as transições e os tokens como bolas pretas dentro dos lugares.

Na figura 4, a representação gráfica de uma RP para um semáforo pode ser observada. Nessa caso, há três lugares, P0, P1 e P3; com três transições T0, T1 e T2, ligadas pelos arcos e com a ficha posicionada no lugar P0.

2.3.2 Conceito de Disparo

Um disparo acontece quando uma transição ou um conjunto de transições é acionado de forma a alterar a marcação de uma RP (SILVA et al., 2017).

No exemplo da figura 4, para que o semáforo vá para o estado amarelo, a transição T_0 é ativada. Para que esse disparo ocorra, deve existir uma ficha na entrada da transição T_0. A figura 5 exemplifica esse disparo.

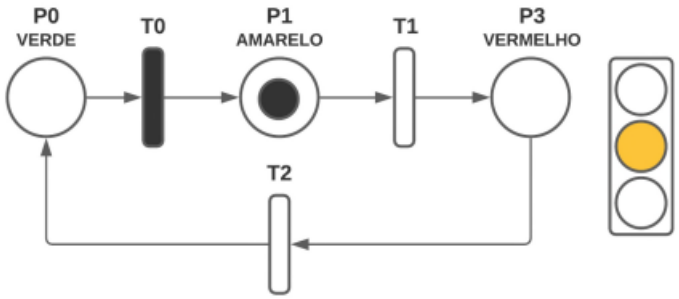

Figura 5 – Disparo da transição T_0 movimentando o token de P_0 para P_1, (PAULA, 2021)

2.3.3 Representação Matemática

A representação matemática de uma RP pode ser definida por uma tupla de 5 elementos $PN = (P, T, A, W, M0)$, onde:

P é o número finito de estados.

T é o número finito de transições.

$A \subseteq (P \times T) \cup (T \times P)$ é o conjunto de arcos direcionados de estado para transição e de transição para estado.

$W : A \rightarrow \{\ 1, 2, 3, ...\ \}$ é a função peso.

$M_0 : P \rightarrow \{\ 0, 1, 2, 3, ...\ \}$ é a marcação inicial.

Com a RP definida, um disparo é caracterizado pela equação (2.1)

$$M_{k+1} = M_k + A^T \times uk \tag{2.1}$$

Onde:

M_{k+1} é o vetor coluna que possui tantos elementos quanto existem lugares na rede.

M_k é o vetor de marcações resultante do disparo.

A^T é a matriz de incidência que representa os arcos de uma rede de Petri.

uk é o vetor de disparos. Possui todos os elementos iguais a zero, exceto o elemento a_k que é igual a 1.

Como exemplo iremos utilizar a figura 4.

Nesse caso:

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

$$M_0 = \begin{vmatrix} 1 \\ 0 \\ 0 \end{vmatrix}$$

Para que o semáforo vá para o estado amarelo, devemos ativar a transição T_0. Então iremos disparar o vetor u_0 que corresponde à essa transição.

$$u_0 = \begin{vmatrix} 1 \\ 0 \\ 0 \end{vmatrix}$$

Com esses valores, a equação (2.1) fica:

$$M_1 = \begin{vmatrix} 1 \\ 0 \\ 0 \end{vmatrix} + \begin{vmatrix} -1 & 0 & 1 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{vmatrix} \times \begin{vmatrix} 1 \\ 0 \\ 0 \end{vmatrix} = \begin{vmatrix} 0 \\ 1 \\ 0 \end{vmatrix}$$

2.3.4 PNML

A Ferramenta de Modelagem para Redes de Petri (PNML) é uma proposta para traduzir as representações matemáticas e gráficas da RP em forma de um arquivo. Nessa proposta, a RP é representada através de um arquivo baseado em XML, onde cada termo da equação (2.1) tem sua representação nesse tipo de arquivo. Na figura 6 temos uma representação simples de uma Rede de Petri e na figura 7 sua representação seu código em PNML.

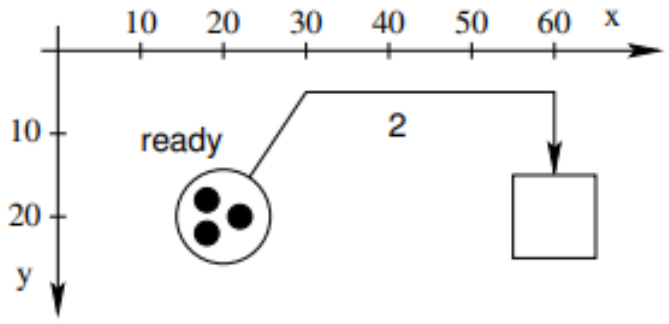

Figura 6 – Representação simples de uma Rede de Lugar/Transição (KINDLER, 2006)

2.4 PNRD

Redes de Petri Elementares inseridas em base de dados RFID (PNRD) é uma estrutura de dados formal a ser armazenada na etiqueta RFID de forma a integrá-la com sistema de controle.(TAVARES; SARAIVA, 2010)

Na PNRD a tag envia suas informações ao leitor, que, ao decodificar essas informações, realiza os cálculos da RP criando um novo vetor de marcação M_k . Caso não haja problema na tag, é feita uma gravação desse vetor M_k. A figura 8 tem uma representação da PNRD.

2.4.1 Exceções

Os disparos da PNRD são padronizados segundo a biblioteca de (SILVA et al., 2017).

Para o módulo da RP, temos:

- *NO_ERROR* – No contexto da verificação de disparo, significa que o disparo descrito não gera erros. Significa que o disparo foi feito e o vetor de marcações foi atualizado;
- *CONDITIONS_ARE_NOT_APPLIED* – Indica que as condições associadas às transições do disparo não estão satisfeitas;

```
<pnml xmlns="http://www.example.org/pnml">
  <net id="n1" type="http://www.example.org/pnml/PTNet">
    <page id="top-level">
      <name>
        <text>An example P/T-net</text>
      </name>
      <place id="p1">
        <graphics>
          <position x="20" y="20"/>
        </graphics>
        <name>
          <text>ready</text>
          <graphics>
            <offset x="-10" y="-8"/>
          </graphics>
        </name>
        <initialMarking>
          <text>3</text>
          <toolspecific tool="PN4all" version="0.1">
            <tokenposition x="-2" y="-2" />
            <tokenposition x="2" y="0" />
            <tokenposition x="-2" y="2" />
          </toolspecific>
        </initialMarking>
      </place>
      <transition id="t1">
        <graphics>
          <position x="60" y="20"/>
        </graphics>
      </transition>
      <arc id="a1" source="p1" target="t1">
        <graphics>
          <position x="30" y="5"/>
          <position x="60" y="5"/>
        </graphics>
        <inscription>
          <text>2</text>
          <graphics>
            <offset x="15" y="-2"/>
          </graphics>
        </inscription>
      </arc>
    </page>
  </net>
</pnml>
```

Figura 7 – Código PNML relativo a figura 6 (KINDLER, 2006)

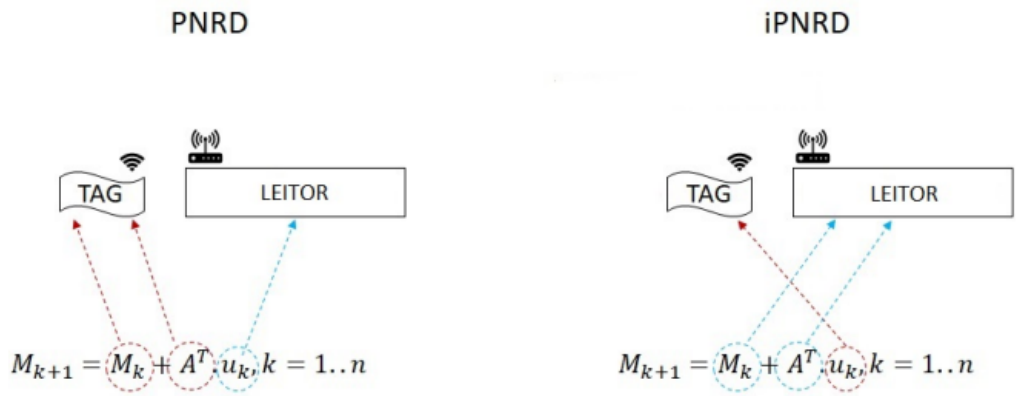

Figura 8 – Associação dos termos da Equação (2.1) com os elementos onde serão armazenados nas abordagens PNRD e iPNRD, (FONSECA et al., 2018)

- *PRODUCE_EXCEPTION* – Indica que o disparo resulta em exceção, ou seja, não existem marcações suficientes nos lugares de entrada para efetuar as transições;
- *NOT_KNOWN* – Quando o programa segue por um fluxo inesperado, usualmente é indicio de um bug;

Com essas condições é possível saber o que ocorreu no processo de gravação/leitura da tag e com isso, definir a melhor abordagem.(PAULA, 2021)

2.4.2 PNRD INVERTIDA (iPNRD)

Em alguns casos, os agentes que requerem as informações são ativos. Neste caso percebe-se que a estrutura clássica da PNRD é incapaz de controlar o comportamento dos mesmos, isso porque agentes ativos possuem comportamento autônomo e seus modelos comportamentais devem ir além do controle de movimentação ao longo dos pontos estratégicos.(PIRES, 2019) Nesse caso, torna-se inviável associar a tag ao agente.

Com isso temos a iPNRD, onde há uma inversão dos locais de armazenamento dos termos contidos na equação (2.1). Na iPNRD, a matriz A^T e o vetor M_k são armazenados no leitor, e o vetor u_k é armazenado na tag. A figura 8 tem uma representação da iPNRD.

2.5 MQTT

O MQTT é um protocolo de mensagens para a Internet das Coisas (IoT). Ele foi projetado como um transporte de mensagens ideal para conectar dispositivos remotos com um pequeno espaço de código e largura de banda de rede mínima.

Em seu funcionamento, publicadores e subscritores se conectam ao um *Broker* e publicam ou estão subscritos em um tópico. Esse *Broker* é um servidor que envia as

mensagens publicadas pelos publicadores para os subscritores. A figura 9 mostra o esquema de funcionamento do protocolo MQTT, onde o publicador envia mensagem para o Broker e o subscritor, subscrito no tópico, recebe a mensagem.

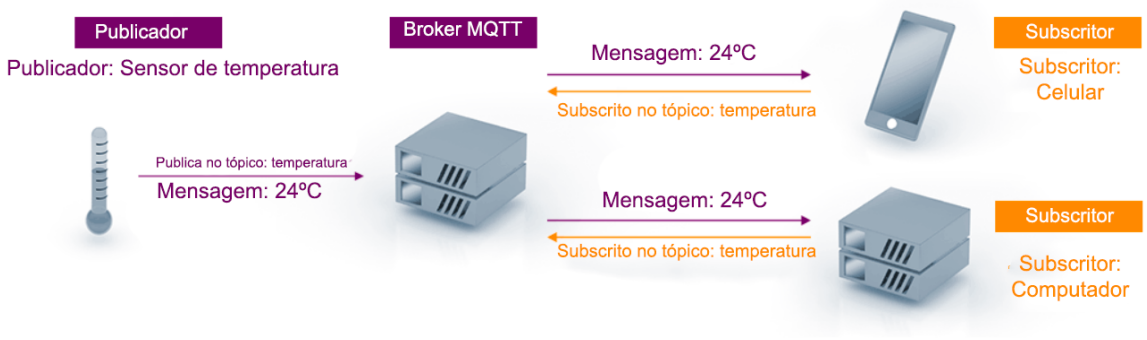

Figura 9 – Esquema de funcionamento de um *Broker* MQTT, (MQTT, 2022), traduzido pelo autor

3 METODOLOGIA

3.1 INTRODUÇÃO

Nesse capítulo será detalhado as ferramentas e abordagens utilizadas na implementação do projeto. As descrições envolvem estudos realizados, opções de projeto e a implementação de Software.

3.1.1 PALMS

O PALMS integra ferramentas de modelagem de RP através de arquivos PNML. Ele converte este arquivo em um arquivo .palms para ser transferido para um Arduino® com leitor RFID e biblioteca PNRD/iPNRD. O PALMS tem uma relação de 1:N entre a RP e os Arduino®.

A conexão do PALMS com os Arduino® é feita através de protocolo MQTT e um esquema de funcionamento do PALMS é exemplificado na figura 10.

Para o funcionamento do PALMS, cada Arduino® deve contar com um Shield Ethernet para conexão na rede e um cartão SD, onde os dados da RP são salvos.

3.2 ESTUDOS INICIAIS

Os estudos iniciais para a implementação do PALMS-MQTT se iniciaram com leituras de artigos sobre as Redes de Petri para melhor compreensão do tema.

Logo após, foram realizados estudos sobre o protocolo MQTT, sua utilização e requisitos necessários para sua utilização. Nessa etapa, foi notado que seria necessário um *Broker* para a comunicação entre publicador e subscritor. O *Broker* escolhido foi o da plataforma <https://www.shiftr.io/>, pois o mesmo tem uma versão grátis que supriu as necessidades dessa pesquisa.

3.2.1 Criando o *Broker*

Para criar o Broker, deve-se acessar o site: <https://www.shiftr.io/> e criar uma conta gratuita.

Após a conta criada, você deve clicar no botão *Deploy Instance*, como visto na figura 12 .

Após clicar no botão, você deve escolher o plano que mais adequa na sua aplicação. No caso desse projeto, foi escolhido o plano gratuito, como exemplificado na figura 13 .

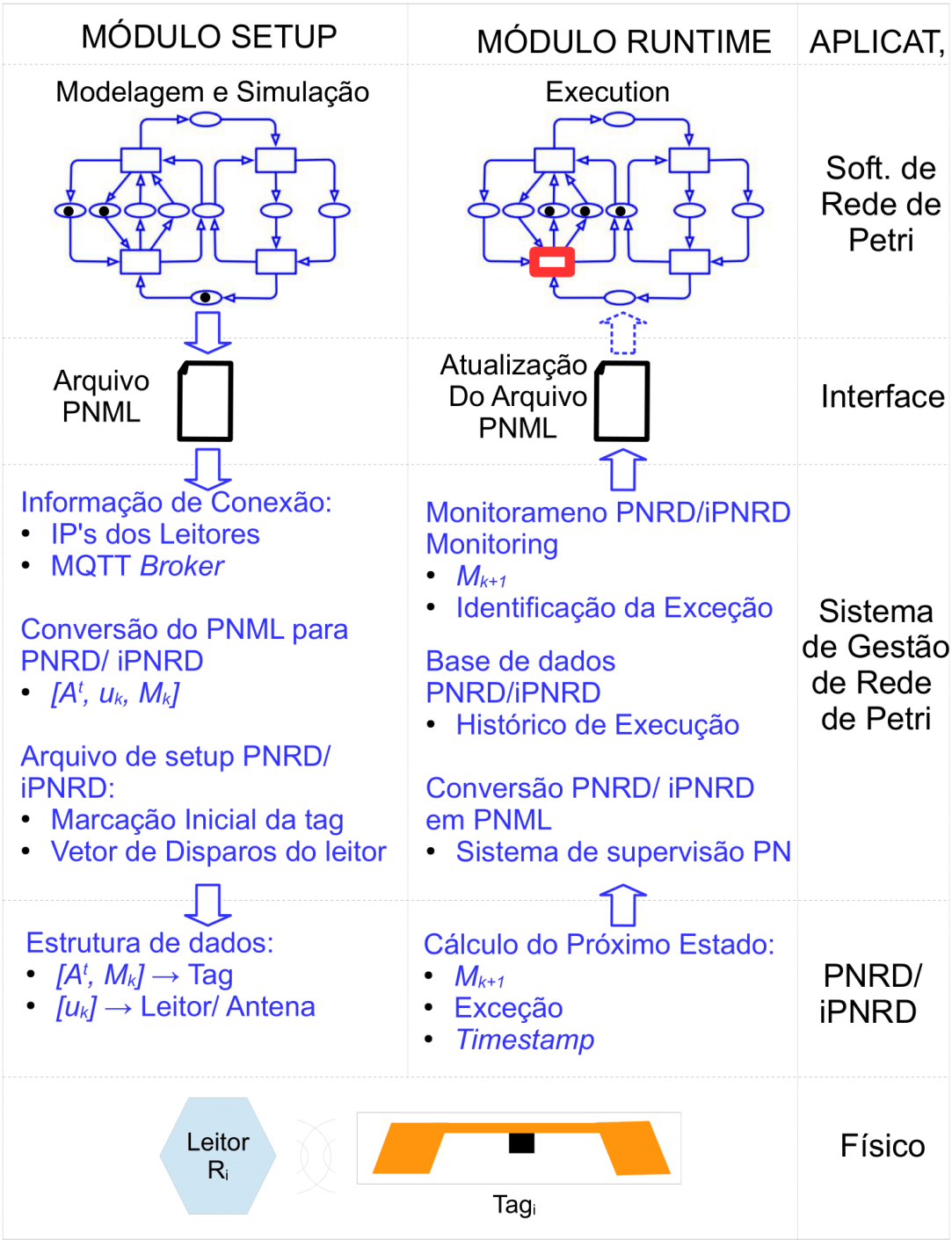

Figura 10 – Esquema de funcionamento do PALMS

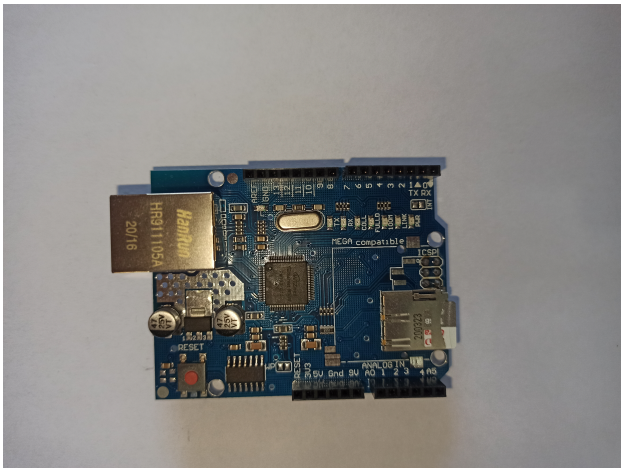

Figura 11 – Shield Ethernet com cartão SD

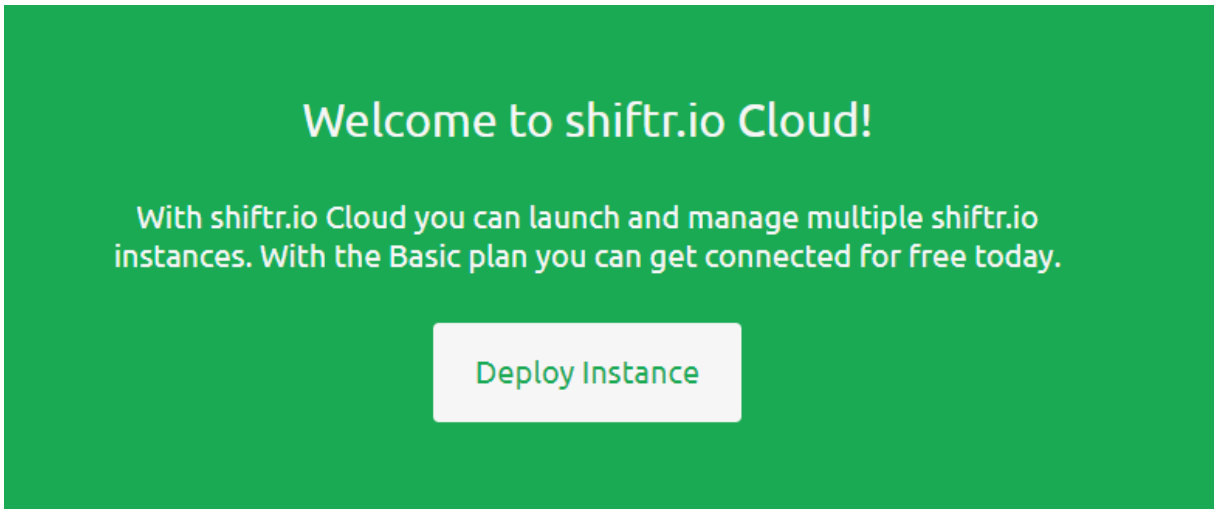

Figura 12 – Criação do *Broker*

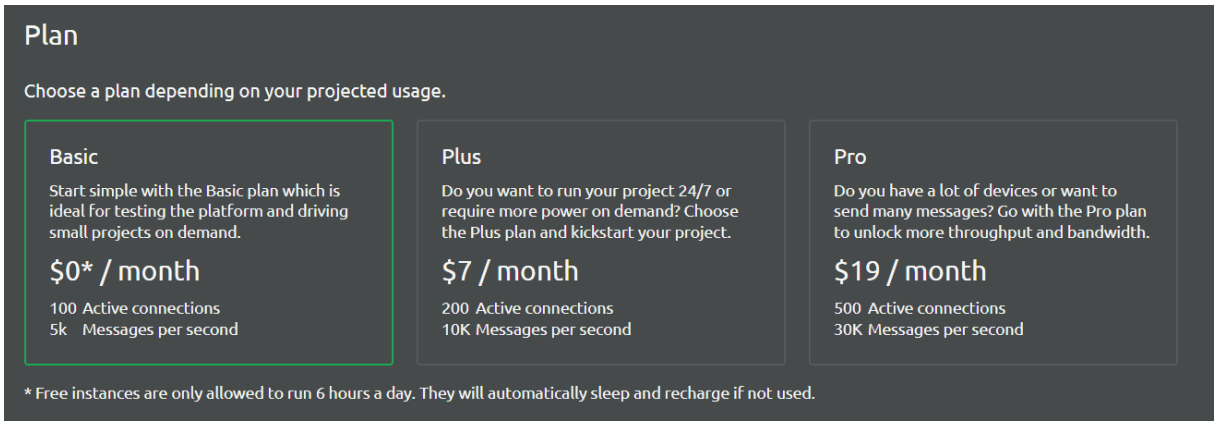

Figura 13 – Planos para criação do *Broker*

Com o plano escolhido, basta preencher os dados relativos à criação, como pode ser visto na figura 14. Ao apertar no botão *Deploy Instance*, seu *Broker* MQTT está pronto para funcionar.

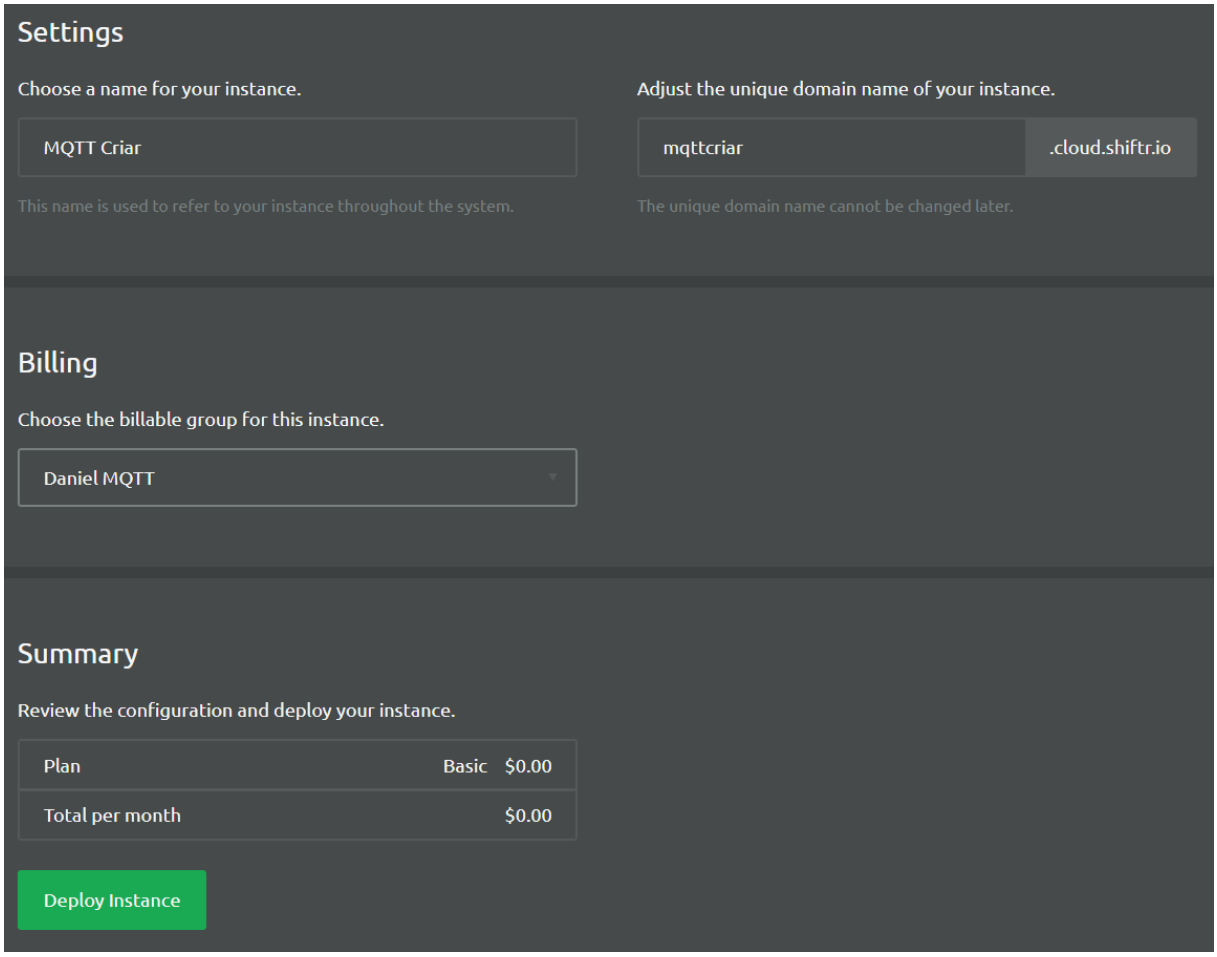

Figura 14 – Dados para criação do *Broker*

O próximo estudo foi sobre o PALMS-FTP, desenvolvido por Roger Carrijo, Gabriel de Brito Silva e o time do MAPL. Essa versão utiliza o protocolo FTP para comunicação entre os Arduinos e o PALMS. Essa versão está disponível em: <https://github.com/MAPL-UFU/palms-ftp>.

3.3 COMUNICAÇÃO PUBLICADOR-SUBSCRITOR

Após a criação do *Broker* gratuito, representado pela figura 15, foi o momento de desenvolver códigos que se comunicassem através do protocolo MQTT.

Foram desenvolvidos publicadores e subscritores que enviavam e recebiam mensagem através do broker.

No estudo do PALMS-FTP, o protocolo FTP comunica através de arquivos .pnrd. Para fazer essa comunicação por MQTT, foram desenvolvidos códigos de publicadores

que eram responsáveis por ler os arquivos .pnrd e enviá-los como mensagem pelo Broker e subscritores que eram responsáveis por receber as mensagens e salvá-las em arquivos no formato .pnrd.

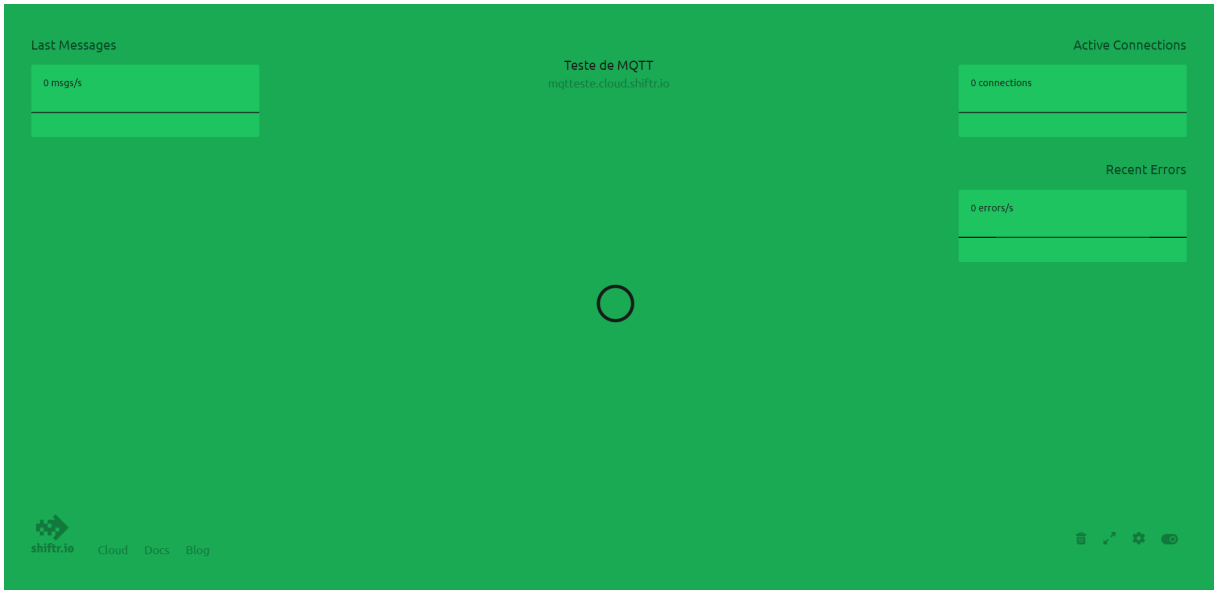

Figura 15 – *Broker* MQTT utilizado no projeto

3.4 COMUNICAÇÃO PYTHON-ARDUINO®

Com os estudos de Python já bem avançados e com os códigos de comunicação funcionando, nessa etapa foi desenvolvida a comunicação com os Arduino®. O Arduino® escolhido inicialmente para essa função, foi o Arduino® UNO com um Shield Ethernet, exemplificado pela figura 16.

Inicialmente foi utilizado o código publicador do Python e um código do Arduino® como subscritor. A partir desses códigos, foram realizadas melhorias no sistema, para que o microcontrolador, ao receber a mensagem do publicador, gravasse a mensagem em um cartão SD. Nesse ponto, foi notado que o Arduino® UNO não tem memória e processamento para realizar a conexão no *Broker*, receber a mensagem enviada a salvá-la no cartão SD. Utilizou-se então o Arduino® Mega com Shield Ethernet. A figura 17 exemplifica esse microcontrolador.

O Arduino® Mega foi capaz de receber as mensagens do *Broker* e salvá-las no cartão SD.

3.5 IMPLEMENTAÇÃO DO PALMS-MQTT

Nessa etapa, foi o momento de atualizar o código do PALMS para comunicação MQTT. Para a comunicação MQTT foram implementadas funções que fazem o sistema se

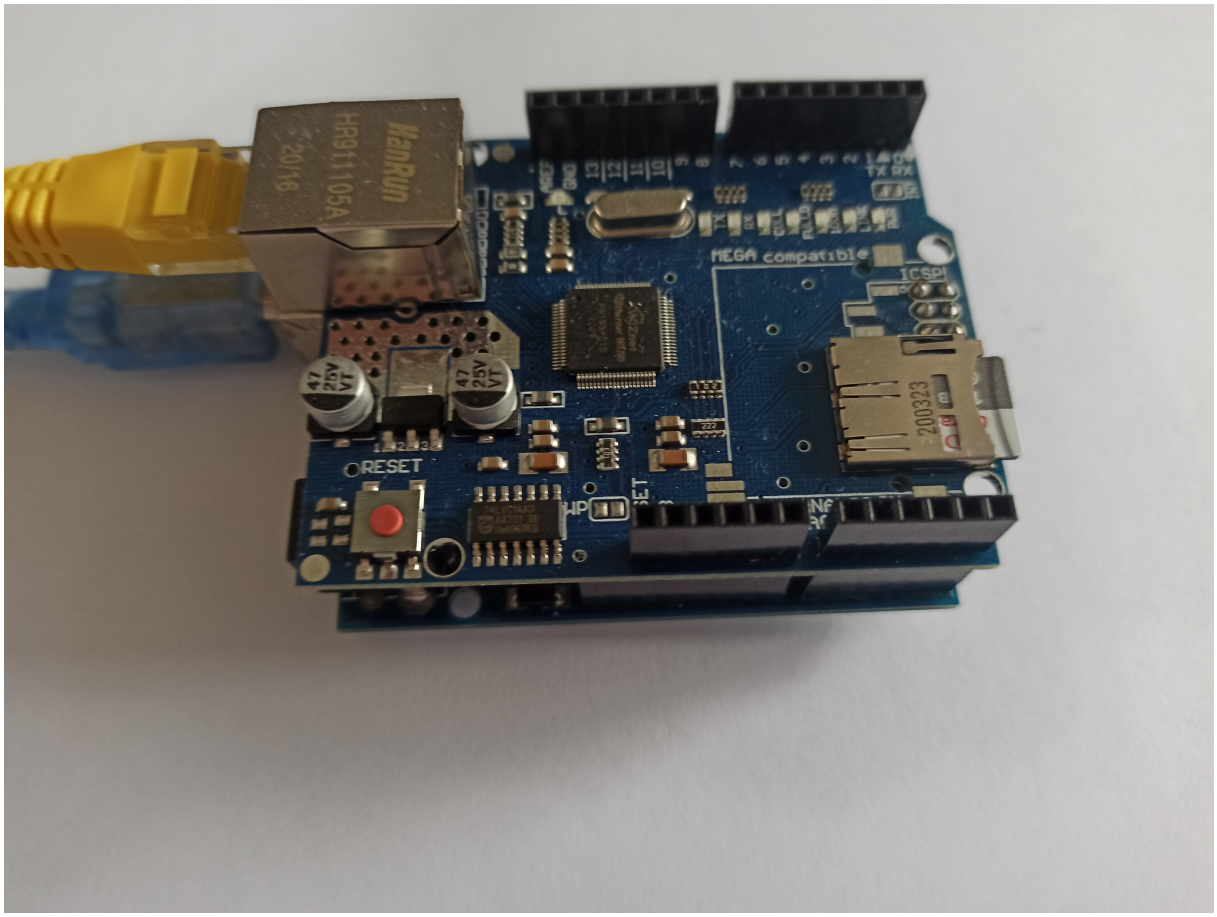

Figura 16 – Arduino® UNO escolhido para comunicação Arduino®-Python

comunicar com esse protocolo.

Foram adicionados ao PALMS conexões com o *Broker*, sistema de publicador, que lê o arquivo .pnrd criado e o envia em uma mensagem através do *Broker* e sistema de subscritor, que lê mensagens recebidas pelos Arduino® sobre as informações da Tag.

Com a implementação do protocolo MQTT ao PALMS, foi o momento de trabalhar nos códigos dos Arduino®.

3.6 IMPLEMENTAÇÃO DO ARDUINO® INICIADOR DE TAGS

Nessa etapa, foi implementado o código para que a Tag seja iniciada com as informações da RP. Esse código foi implementado com base na biblioteca de (SILVA et al., 2017). Como o Arduino® não tem gerenciamento de memória dinâmico, foi usado um cartão SD para salvar as constantes alterações dos termos da RP. Como este é um projeto com caráter didático e acadêmico, a escolha do uso desse microcontrolador não foi um problema.

Para a inicialização da Tag foi utilizado um Arduino® Mega conectado a um leitor PN532 RFID para gravação dos dados. A figura 18 representa o leitor e a figura 19 a

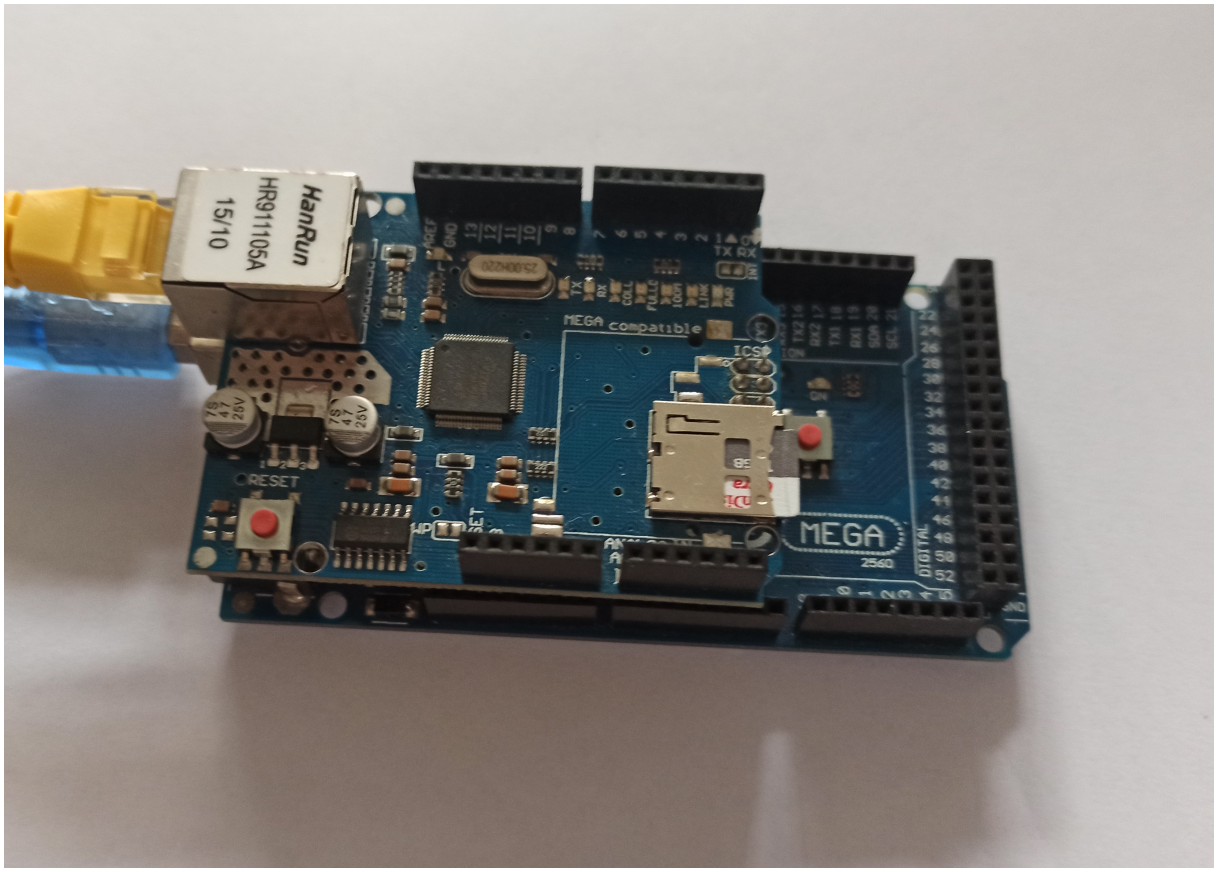

Figura 17 – Arduino® Mega com Shield Ethernet utilizado para comunicação Arduino® - Python

montagem realizada.

O código implementado nessa etapa faz a conexão do Arduino® ao *Broker* MQTT, lê a mensagem publicada pelo PALMS e salva a mesma no cartão SD em um arquivo .pnrd. Depois, lê o arquivo salvo no formato de texto e salva os valores do Lugar e Transição em variáveis de números inteiros e os valores dos Vetores de Estados e da Matriz de Incidência em variáveis de vetores de números inteiros.

Com esses valores armazenados nas variáveis designadas, a PNRD é iniciada no Arduino® e seus valores são salvos nas Tags RFID.

3.7 IMPLEMENTAÇÃO DO ARDUINO® LEITOR

Nessa etapa, foi implementado o código para que a Tag seja lida pelos leitores de um Arduino® e, através da leitura, o disparo resultado da leitura da Tag é salvo para ser enviado para o PALMS. Essa implementação também foi baseada na biblioteca de (SILVA et al., 2017)

Para o leitor da Tag foi utilizado um Arduino® Mega conectado a três placas PN532 RFID responsáveis pelas leituras das Tags. O exemplo da montagem está exemplificado

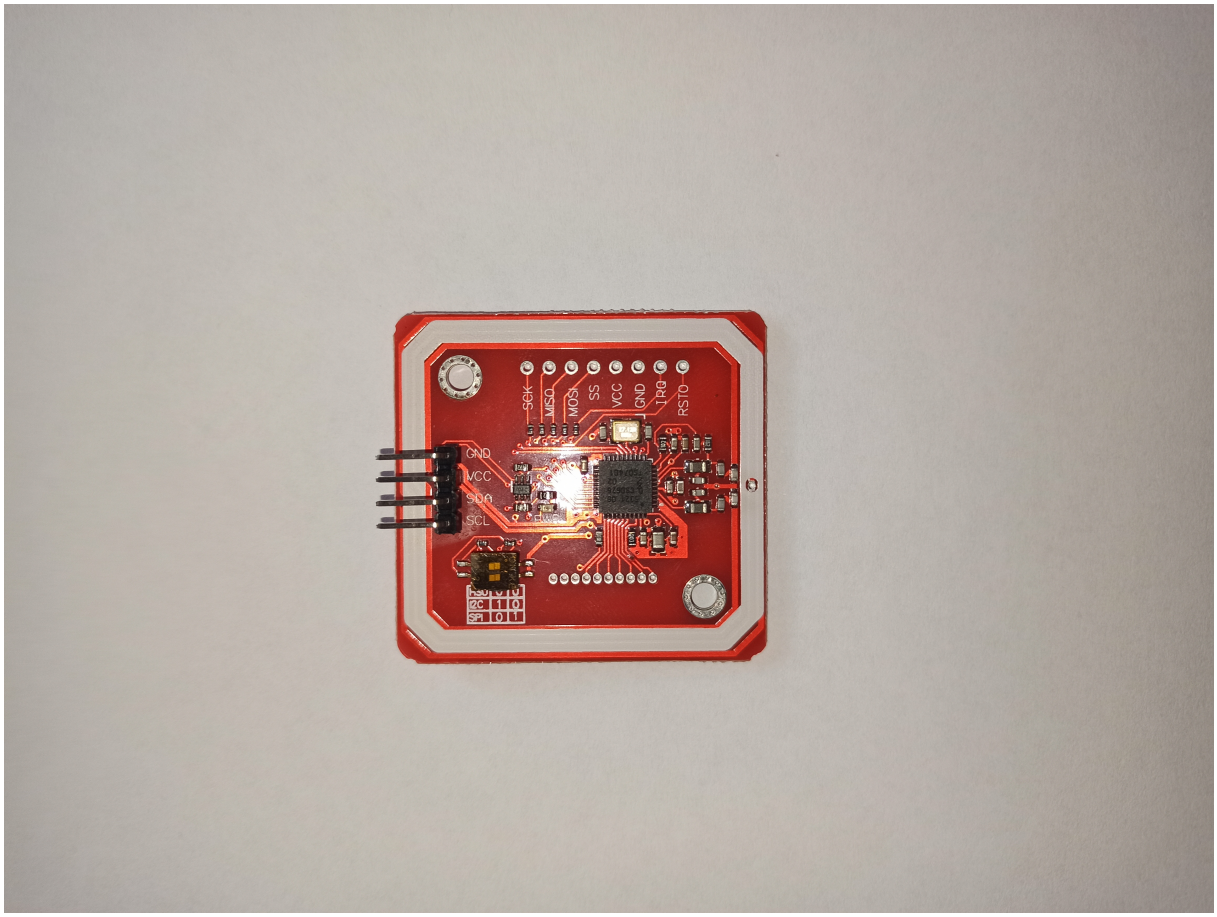

Figura 18 – Leitor PN532 RFID utilizado

na figura 20. Esse tipo de montagem também pode ser vista em (SILVA et al., 2017).

Assim como o código do Arduino Iniciador de Tags, o Arduino Leitor também recebe dados do PALMS e salva esses dados no cartão SD. Os dados no cartão SD são armazenados em variáveis para as PNRDs serem iniciadas. Para cada leitor, uma PNRD é iniciada.

Os leitores são responsáveis por ler os dados salvos nas Tags RFID e, através dos cálculos da RP dispara um resultado que é salvo em uma variável. Quando essa informação é requisitada pelo PALMS, ela é enviada através do protocolo MQTT.

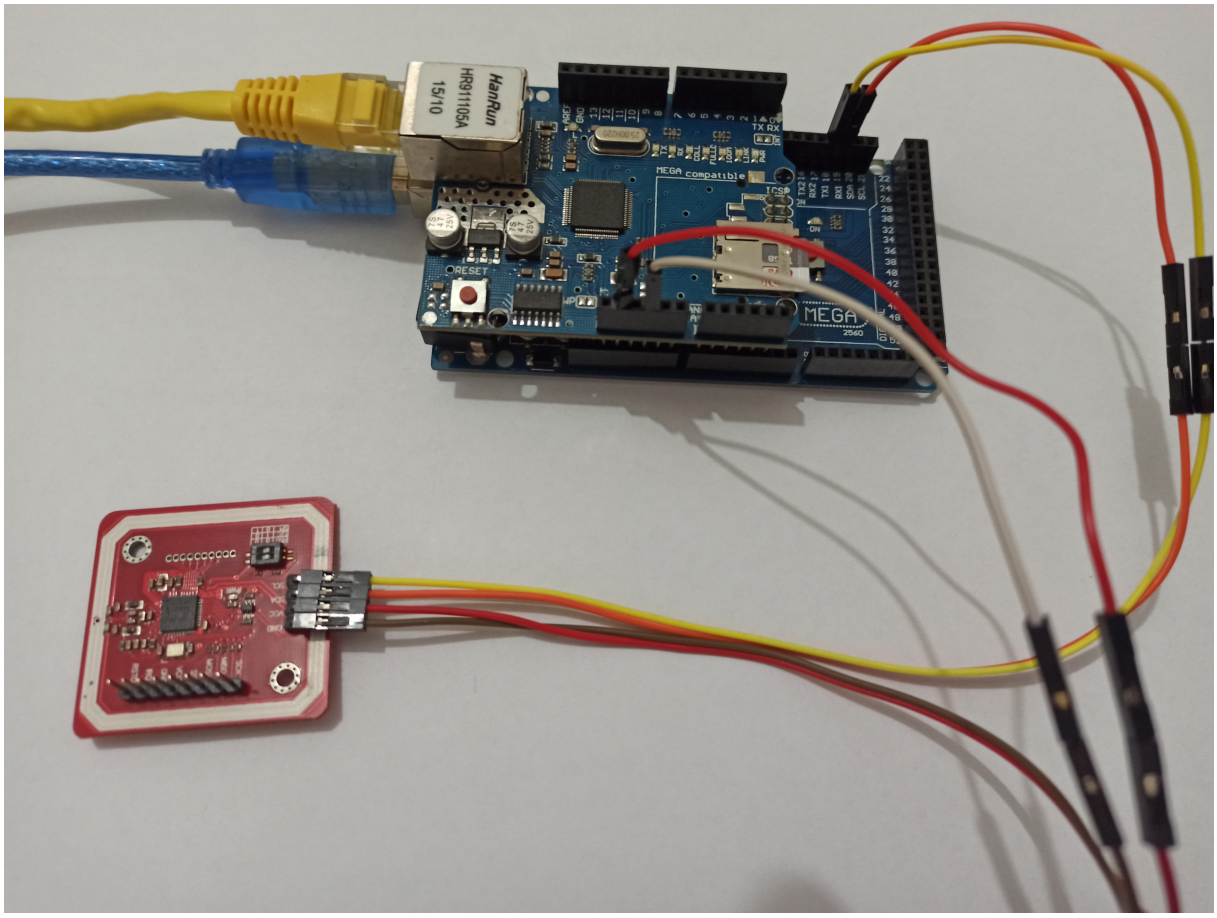

Figura 19 – Montagem realizada para inicialização da Tag

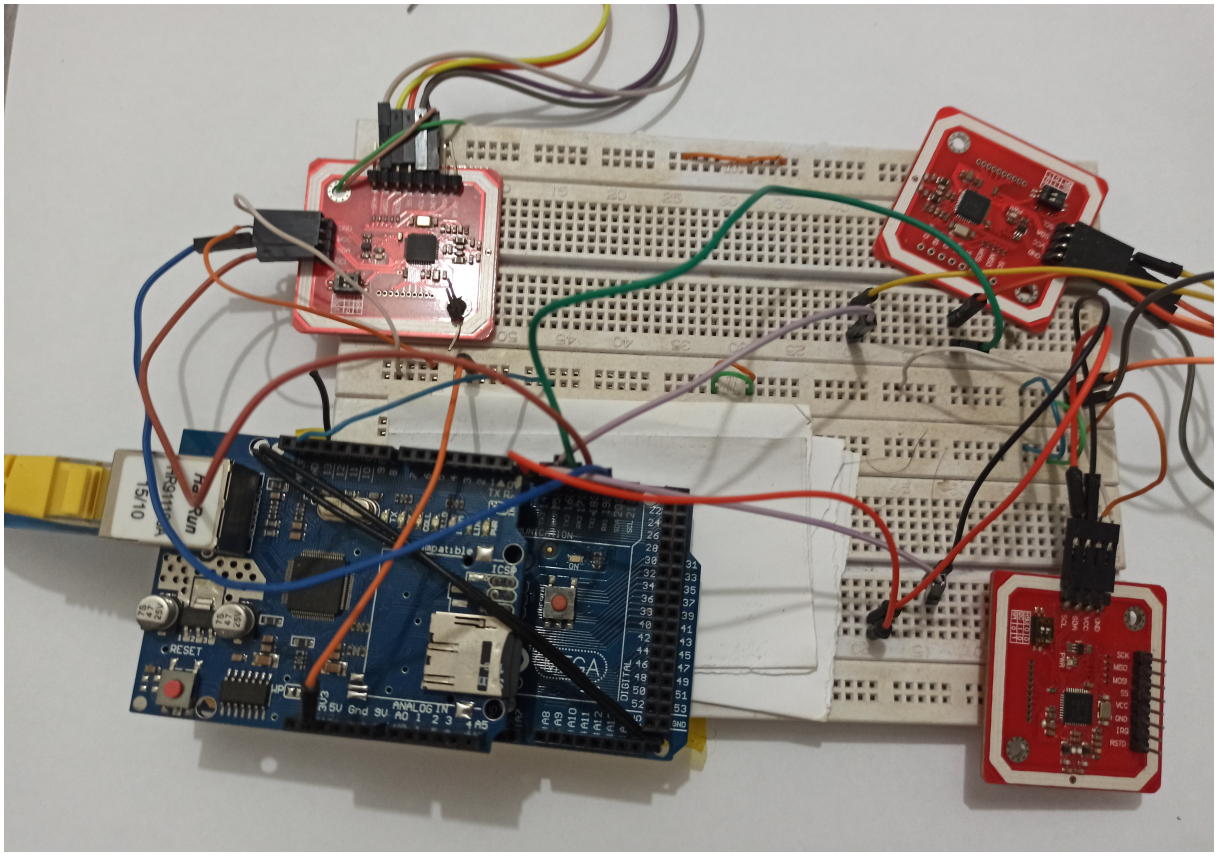

Figura 20 – Montagem realizada para leitura da Tag

4 RESULTADOS E DISCUSSÃO

Nesse capítulo, serão abordados os resultados da implementação do PALMS-MQTT e o modo de utilização desse sistema.

Os códigos implementados para o PALMS-MQTT estão disponíveis em <https://github.com/MAPL-UFU/PALMS-MQTT>.

O PALMS-MQTT tem dois modos, o modo SETUP e o modo RUNTIME. Cada um desses modos será expecificado a seguir.

4.1 MODO SETUP

No modo SETUP, deve-se carregar a PNRD através do arquivo PNML correspondente. O PALMS extrai a Matriz de Incidência e o Vetor de Estados do arquivo carregado. A figura 21 exemplifica o modo inicial do PALMS.

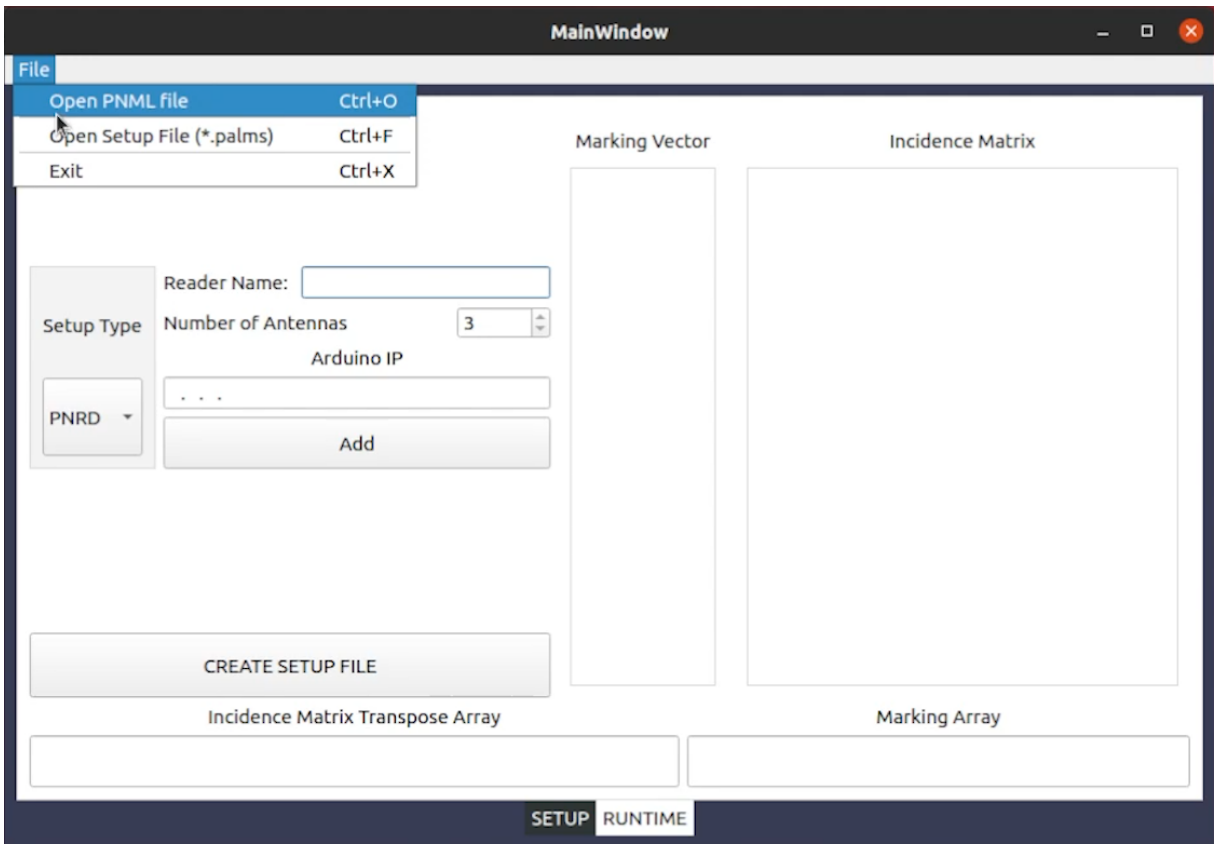

Figura 21 – Carregando a PNRD no PALMS

Após a PNRD carregada, deve-se adicionar os IPs dos Arduino® usados para a leitura da Tag e a quantidade de antenas de cada microcontrolador. Nos testes realizados, cada Arduino® Mega é capaz de suportar até três leitores. Após as informações de IPs

e antenas serem adicionadas, ao pressionar o botão CREATE SETUP FILE, o arquivo setup.palms relativo a essa PNML e a essas informações é criado. A figura 22 exemplifica o modo SETUP assim que o arquivo setup é criado.

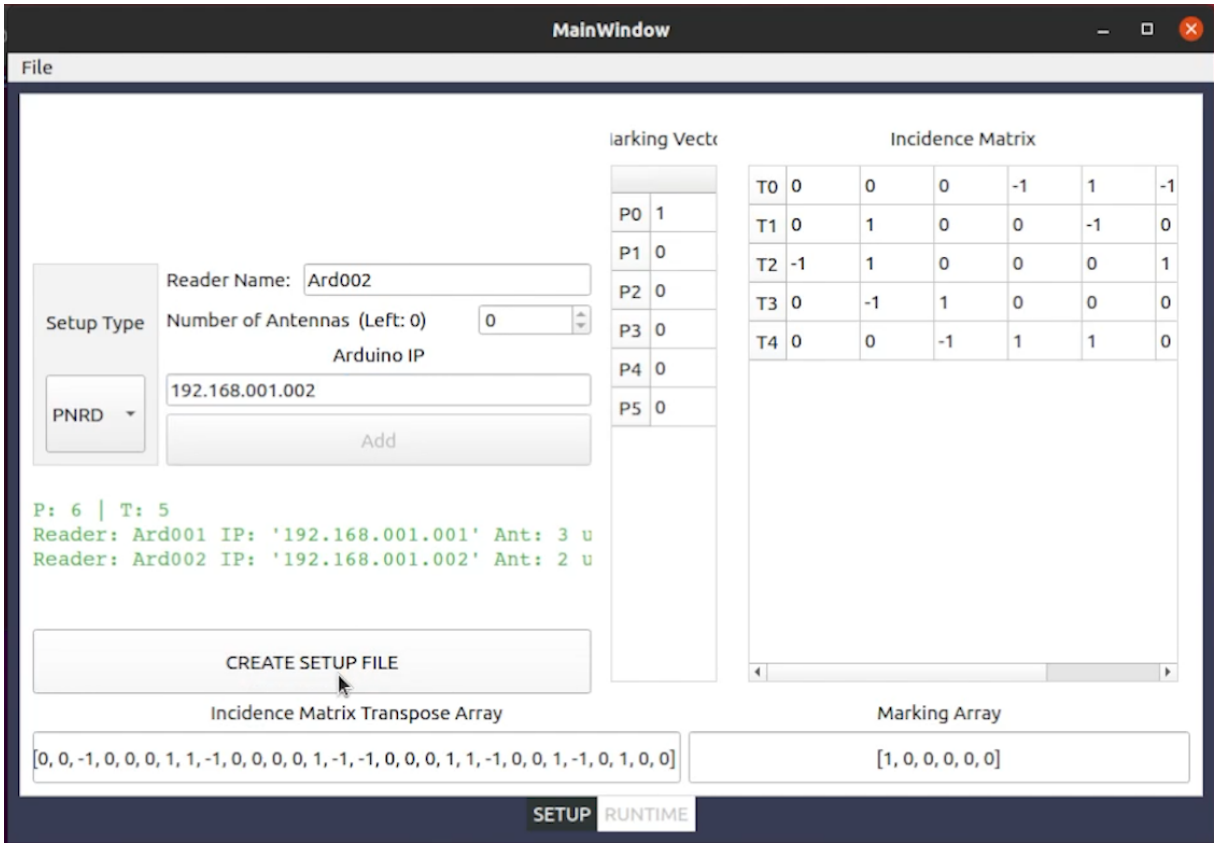

Figura 22 – Modo SETUP do PALMS na criação do arquivo setup

4.2 MODO RUNTIME

Abrindo o arquivo setup.palms, é aberto o modo RUNTIME. No modo RUNTIME, como mostrado na figura 23, estão as informações da PNRD, dos IPs e antenas dos Arduino® adicionadas no modo Setup.

Pressionando o botão TRANSFER INITIAL PNRD, a PNRD inicial é enviada via MQTT para o Arduino® que irá salvar os dados na Tag. A figura 24 mostra o Arduino recebendo os dados do PALMS e salvando esses dados no cartão SD. A figura 25 mostra o Arduino® salvando os dados na Tag.

Pressionando o botão TRANSFER PNRD SETUP, o PALMS envia para o Arduino® responsável pela leitura a Tag, os dados relacionados a PNRD. A figura 26 mostra o microcontrolador recebendo os dados, salvando no cartão SD e lendo a Tag RFID.

Pressionando o botão GET RUNTIME INFO, o PALMS envia um comando para o Arduino® enviar os dados relacionados a leitura da Tag. A figura 27 mostra o PALMS

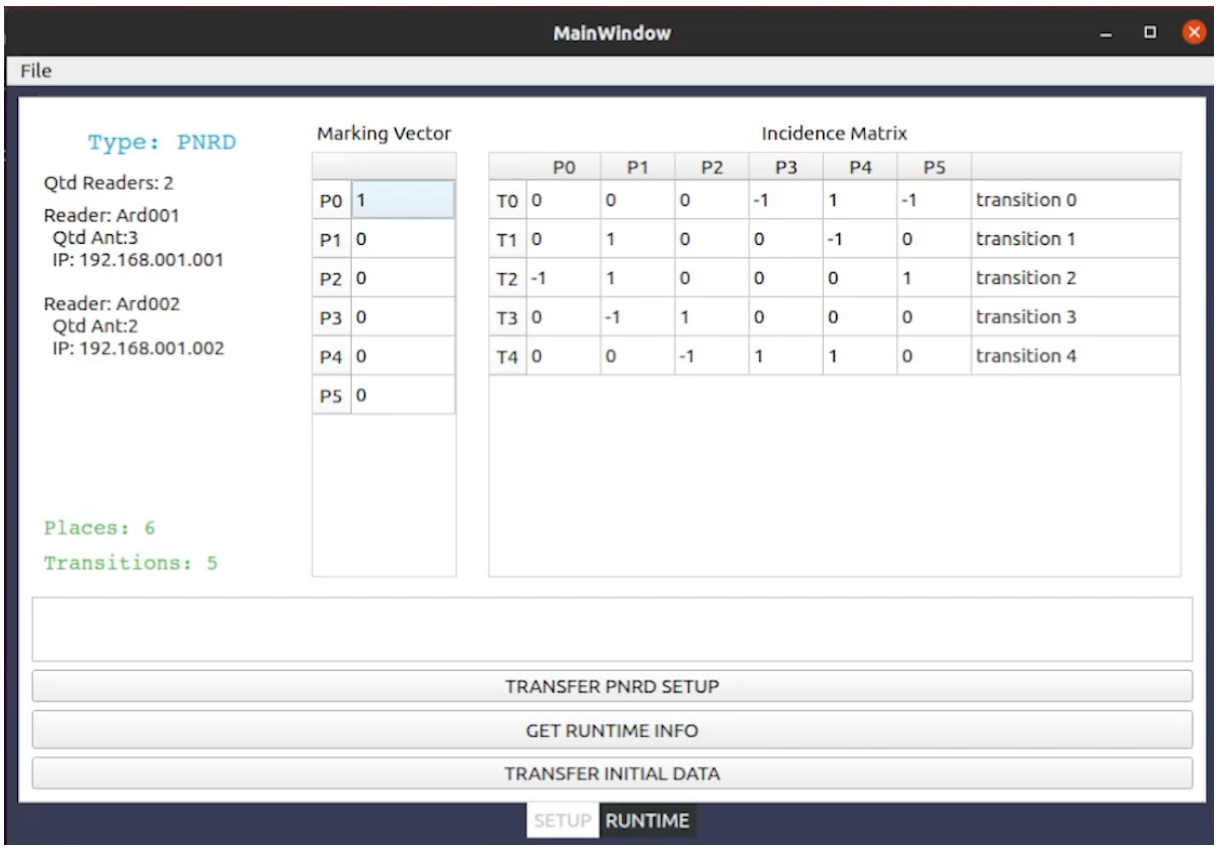

Figura 23 – Modo RUNTIME do PALMS

```
Lendo o cartão SD
Places = 6
Transitions = 5
Vetor = {1,0,0,0,0,0}
Matriz = {0,0,-1,0,0,0,1,1,-1,0,0,0,0,1,-1,-1,0,0,0,1,1,-1,0,0,1,-1,0,1,0,0}
Places = 6
Transitions = 5
Vetor de estados = {1,0,0,0,0,0,}
Matriz de incidência = {0,0,-1,0,0,0,1,1,-1,0,0,0,0,1,-1,-1,0,0,0,1,1,-1,0,0,1,-1,0,1,0,0,}
Found chip PN532
Firmware ver. 1.6

Machine 1: Initial tag recording.Places = 6
Transitions = 5
Place an tag to be configurated.
```

Figura 24 – Arduino® Iniciador de Tags salvando os dados e lendo o cartão SD

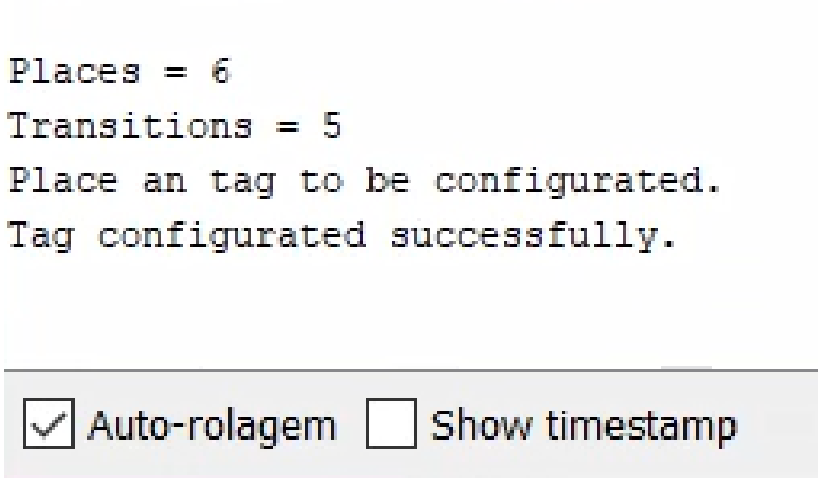

Figura 25 – Tag corretamente configurada com os dados

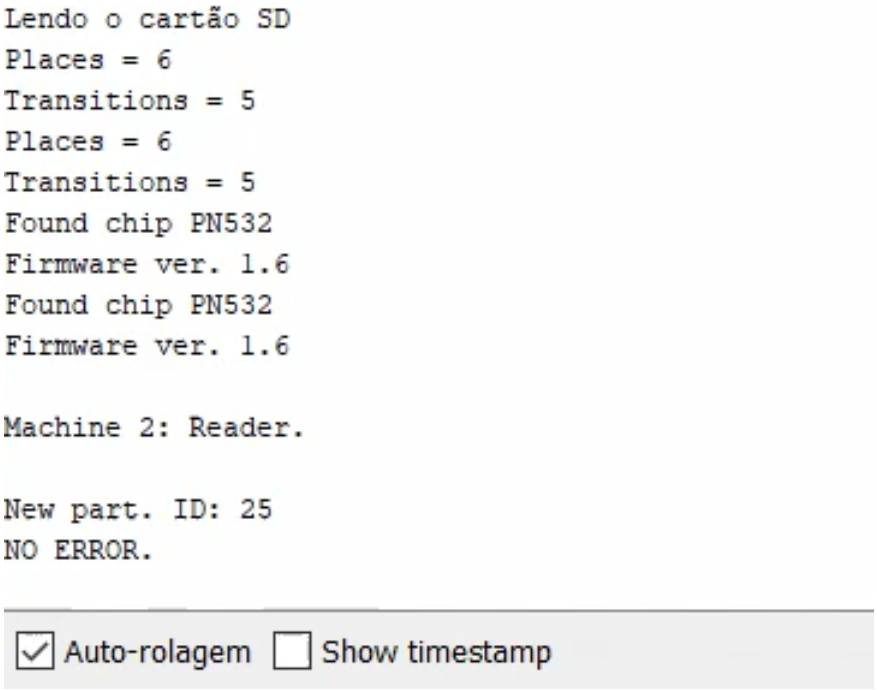

Figura 26 – Lendo a Tag RDIF

recebendo um disparo NO ERROR e a figura 28 mostra o PALMS recebendo um disparo CONDITIONS ARE NOT APLIED.

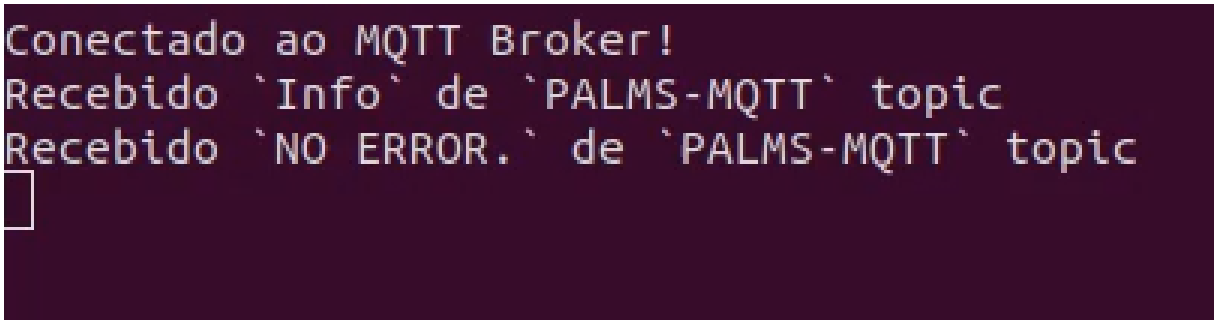

Figura 27 – PALMS recebendo informação NO ERROR da Tag

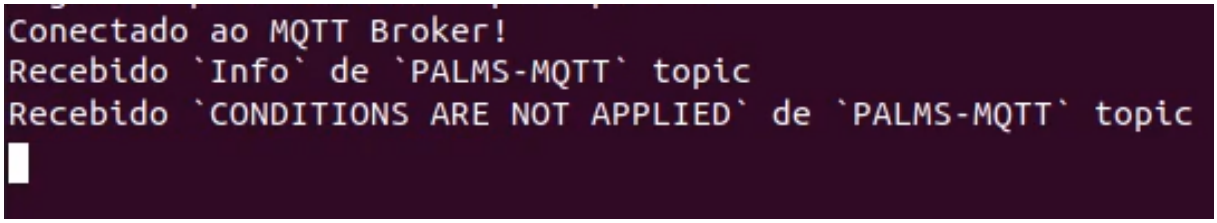

Figura 28 – PALMS recebendo informação CONDITIONS ARE NOT APLIED da Tag

5 CONCLUSÃO

O projeto desenvolvido e apresentado nesse trabalho atingiu os objetivos esperados e definidos. A implementação do protocolo MQTT no PALMS garantiu ao sistema uma característica de IoT, com comunicação em Nuvem e em tempo real.

O Arduino® Mega se mostrou capaz de suportar as necessidades do projeto permitindo a interação com até três leitores, e a escolha do protocolo MQTT se mostrou acertada, pois há uma infinidade de Broker, de gratuitos a pagos, que podem ser escalonados de acordo com a necessidade do sistema.

Com o PALMS é possível monitorar várias PNRD/iPNRD ao mesmo tempo, incluindo identificação de exceções.

Outra inovação do PALMS-MQTT é o fato de que todas as variáveis iniciais relacionadas a RP são enviadas através do Broker para o Arduino®. Na versão arterior, as matrizes e vetores da RP deviam estar no código dos microcontroladores. Nessa versão eles são recebidos via MQTT e salvos no cartão SD.

A integração do PALMS com o protocolo MQTT e o uso de cartão SD traz ao processo que o usa uma grande robustez e autonomia. Sendo necessária apenas uma fonte de energia e de conexão com a internet para os Arduino® , o PALMS gera uma relacão 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 cartão SD, mantendo o sistema ainda ativo.

Esse trabalho possui um caráter didático e se trata de um protótipo. O PALMS é um software livre e pode ser alterado para usos industriais e acadêmicos.

6 TRABALHOS FUTUROS

Visando a melhoria e escalabilidade da solução, os trabalhos futuros poderiam:

6.1 IMPLEMENTAÇÃO DE UMA VERSÃO QUE UTILIZA TANTO FTP QUANTO MQTT

Uma versão futura do PALMS, onde o mesmo trabalha com o MQTT em tempo real e o FTP como *backup*, pode ser implementada. As linhas de código que utilizam a versão FTP não foram apagadas e, necessitam apenas de uma revisão e finalização para total funcionamento.

Essa nova versão deveria fazer um teste de conexão dos dois protocolos e, se um dos dois falhar, usar o que está disponível. Caso ambos os protocolos apresentem conexão com sucesso, usar o protocolo MQTT, que se comunica em tempo real, como o protocolo primário o o FTP, que envia arquivos, como backup.

6.2 IMPLEMENTAÇÃO DO PALMS-MQTT EM UMA *BLOCKCHAIN*

Para que todos os dados da RP e todas as alterações das Tags e no cartão SD fiquem salvas e tenham maior confiabilidade, uma *Blockchain* acoplada ao sistema do PALMS-MQTT poderia ser implementada.

6.3 IMPLEMENTAR UMA VERSÃO QUE PERMITA ALTERAÇÕES AO LONGO DA EXECUÇÃO

Uma versão futura do PALMS-MQTT, que possa permitir que o IP do broker possa ser alterado através da interface do programa. Pode-se também implementar um código para que os Arduino® Leitor e Iniciador de Tags possam realizar as duas funções com um único código, não sendo mais necessários códigos distintos para os microcontroladores.

6.4 HOMOLOGAÇÃO E CERTIFICAÇÕES

Para que o PALMS possa ser utilizado em um ambiente industrial, deve-se utilizar um sistema homologado pelo órgão fiscalizador responsável pelo local e certificações de segurança relacionadas ao tipo de atividade realizada pela indústria.

Podemos ter uma versão do PALMS-MQTT montada com equipamentos homologados e com certificação de segurança.

REFERÊNCIAS

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. Citado nas páginas 7 e 20.

KINDLER, E. Concepts, status, and future directions. University of Paderborn, 2006. Citado nas páginas 7, 18 e 19.

MQTT. **MQTT Web Page**. 2022. Acessado em: 01/08/2022. Disponível em: <http://mqtt.org/>. Citado nas páginas 7 e 21.

MURATA, T. Petri nets: Properties, analysis and applications. **Proceedings of the IEEE**, IEEE, v. 77, n. 4, p. 541–580, 1989. Citado na página 15.

PAULA, R. H. C. d. Integração da pnrd com banco de dados distribuído hyperledger sawtooth. Universidade Federal de Uberlândia, 2021. Citado nas páginas 7, 13, 15, 16 e 20.

PIRES, A. R. B. Sistema ciber-físico baseado na integração pnrd e ipnrd utilizando cloud conputing. Universidade Federal de Uberlândia, 2019. Citado nas páginas 12 e 20.

SILVA, C. E. A. d. et al. Desenvolvimento de biblioteca para aplicações de pnrd e pnrd invertida embarcadas em arduino. Universidade Federal de Uberlândia, 2017. Citado nas páginas 13, 16, 18, 27, 28 e 29.

SUNDMAEKER, H. et al. Vision and challenges for realising the internet of things. **Cluster of European Research Projects on the Internet of Things, European Commision**, v. 3, n. 3, p. 34–36, 2010. Citado na página 12.

TAURION, C. **Cloud computing-computação em nuvem**. [S.l.]: Brasport, 2009. Citado na página 12.

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. Citado na página 18.

A APÊNDICE - PROGRAMA PUBLICADOR EM PYTHON

Programa em Python para o Publicador responsável por ler os arquivos .pnrd e enviá-los como mensagem.

```
import random
import time
import sys
from paho.mqtt import client as mqtt_client


broker = 'mqtteste.cloud.shiftr.io'
port = 1883
topic = "MENSAGEM"
# generate client ID with pub prefix randomly
client_id = f'python-mqtt-{random.randint(0, 100)}'
username = 'mqtteste'
password = 'testes'
flag = 0
msg_flag=0

def connect_mqtt() -> mqtt_client:
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("Conectado ao MQTT Broker!")
        else:
            print("Falha na conex o , return code %d\n", rc)

    client = mqtt_client.Client(client_id)
    client.username_pw_set(username, password)
    client.on_connect = on_connect
    client.connect(broker, port)
    return client


def subscribe(client: mqtt_client):
    def on_message(client, userdata, msg):
        print(f"Recebido `{msg.payload.decode()}` de `{msg.topic}` topic"
```

```
        msg_flag=msg.payload.decode()
        print(msg_flag)
    global msg_flag
    client.subscribe(topic)
    client.on_message = on_message
    if msg_flag == '-1':
        client.disconect()

def publish(client):
    while True:
        time.sleep(1)
        arquivo = open('arq02.txt','r')
        msg = arquivo.read()
        result = =client.publish(topic, msg)
        with open(f'pnrd_initTag.pnrd', 'r') as text_file:
        print(text_file.read())
        status = result[0]
        if status == 0:
           msg = '-1'
           status = 0
        else:
            print(f"Failed to send message to topic {topic}")
        if msg == '-1':
            break

def run():
    client = connect_mqtt()
    global msg_flag
    flag = int (input("Digite 0 para subs ou 1 para pub: "))
    if flag == 0:
        subscribe(client)
        client.loop_forever()
        if msg_flag == '-1':
            client.loop_stop()
    elif flag == 1:
        publish(client)
        client.loop_start()
    else:
        print ("Comando inv lido")
```

```
if __name__ == '__main__':
    run()
```

B APÊNDICE - PROGRAMA SUBSCRITOR EM PYTHON

Programa em Python para o Subscritor, responsável por receber as mensagens e salvá-las em arquivos no formato .pnrd

```
import random
import time
import sys
from paho.mqtt import client as mqtt_client


broker = 'mqtteste.cloud.shiftr.io'
port = 1883
topic = "MENSAGEM"
# generate client ID with pub prefix randomly
client_id = f'python-mqtt-{random.randint(0, 100)}'
username = 'mqtteste'
password = 'testes'
flag = 0
msg_flag=0
with open('arq01.txt', 'w') as arquivo:
arquivo.close()

def connect_mqtt() -> mqtt_client:
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("Conectado ao MQTT Broker!")
        else:
            print("Falha na conex o , return code %d\n", rc)

    client = mqtt_client.Client(client_id)
    client.username_pw_set(username, password)
    client.on_connect = on_connect
    client.connect(broker, port)
    return client


def subscribe(client: mqtt_client):
```

```
    def on_message(client, userdata, msg):
        print(f"Recebido `{msg.payload.decode()}` de `{msg.topic}` topic"
        msg_flag=msg.payload.decode()
        print(msg_flag)
        arquivo = open('arq01.pnrd', 'a')
        if msg_flag == '':
        arquivo.close()
        else:
        arquivo.write(msg_flag)
        arquivo.write('\n')

    global msg_flag
    client.subscribe(topic)
    client.on_message = on_message
    if msg_flag == '-1':
        client.disconect()

def publish(client):
    while True:
        time.sleep(1)
        msg = str (input("Digite a mensagem a ser enviada(-1 para sair) "
        result = client.publish(topic, msg)
        # result: [0, 1]
        status = result[0]
        if status == 0:
            print(f"Send `{msg}` to topic `{topic}`")
        else:
            print(f"Failed to send message to topic {topic}")
        if msg == '-1':
            break

def run():
    client = connect_mqtt()
    global msg_flag
    flag = int (input("Digite 0 para subs ou 1 para pub: "))
    if flag == 0:
        subscribe(client)
        client.loop_forever()
        if msg_flag == '-1':
```

```
            client.loop_stop()
    elif flag == 1:
        publish(client)
        client.loop_start()
    else:
        print ("Comando inv lido")



if __name__ == '__main__':
    run()
```

C APÊNDICE - PROGRAMA DO ARDUINO® INICIADOR DE TAGS

Programa do Arduino® Iniciador de Tags no Arduino® Mega.

```
#include <MQTT.h>
#include <Ethernet.h>
#include <SPI.h>
#include "SdFat.h"
#include <Pn532NfcReader.h>
#include <PN532_HSU.h>



byte mac[] = {0xE4, 0x83, 0x26, 0xC4, 0x9a, 0xED};
byte ip[] = {10, 0, 208, 12};
byte gate[] = {10 , 0, 208, 1};
byte masc[] = {255, 255, 252, 0};
byte dns[] = {200, 19, 146, 201};


#define SD_CS_PIN 4
File myFile;
SdFat SD;

unsigned long lastMillis = 0;
String total_msg;

uint8_t int_places;
uint8_t int_transitions;

String places;
String transitions;
String matriz;
String vetor;

int n;
```

```
int j;
char temp;
String vetor_temp;
String matriz_temp;

int k;
int cont;
int l;

String msg;
String valores;
int contador;


EthernetClient net;
MQTTClient client_cloud;

//Rotines related with the configuration of the RFID reader PN532
PN532_HSU pn532hsu(Serial1);
NfcAdapter nfc = NfcAdapter(pn532hsu);

//Creation of the reader and PNRD objects
Pn532NfcReader* reader = new Pn532NfcReader(&nfc);
Pnrd pnrd = Pnrd(reader,int_places,int_transitions);//leitor, no estados


void setup() {

    Serial.begin(115200);

  // Open serial communications and wait for port to open:
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }


  Serial.print("Initializing SD card...");

  if (!SD.begin(SD_CS_PIN)) {
```

```
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");


// Configura  es para o uso do MQTT
    Ethernet.begin(mac, ip, dns, gate, masc);
    client_cloud.begin("mqtteste.cloud.shiftr.io", net);
    client_cloud.onMessage(messageReceived);
    connect();



//////LENDO CARTO SD///////

    Serial.println("Lendo o cart o SD");
    contador = 0;
    myFile = SD.open("PNRDINFO.pnrd");
    if (myFile) {
      while (myFile.available()) {
        String list = myFile.readStringUntil('\n');
        contador++;
        if (contador == 1){
          places = list;
          Serial.println("Places = " + places);
        }
        if (contador == 2){
          transitions = list;
          Serial.println("Transitions = " + transitions);
        }
        if (contador == 3){
          vetor = list;
          Serial.println("Vetor = " + vetor);
        }
        if (contador == 4){
          matriz = list;
          Serial.println("Matriz = " + matriz);
        }
```

```
        }
        // close the file:
        myFile.close();
      } else {
        // if the file didn't open, print an error:
        Serial.println("error opening test.txt");
      }



/////DESCOBRINDO VALORES DE PLACE E TRANSITIONS///////

    int_places = places.toInt();
    int_transitions = transitions.toInt();
    Serial.print("Places = " );
    Serial.println(int_places);
    Serial.print("Transitions = " );
    Serial.println(int_transitions);

//////DESCOBRINDO VALOR DO VETOR DE ESTADOS/////////

    n = vetor.length();
    j = 0;
    for (int  i = 1; i < (n-1) ; i = i+2){
      vetor_temp += vetor.charAt(i);
    }
    j = vetor_temp.length();

    uint16_t StartingTokenVector[j];

    for (int  i = 0; i < j ; i++){
     temp = vetor_temp.charAt(i);
     StartingTokenVector[i] =(temp-48);
    }
    Serial.print ("Vetor de estados = {");
    for (int  i = 0; i < j ; i++){
     Serial.print(StartingTokenVector[i]);
     Serial.print(",");
    }
```

```
    Serial.println("}");



//////DESCOBRINDO VALOR DA MATRIZ DE INCIDENCIA/////////

    n = matriz.length();
    j = 0;
    for (int  i = 1; i < (n-1) ; i++){
     temp = matriz.charAt(i);
     if (temp == 44){
       i++;
     }
     if(temp == 45){
         matriz_temp += matriz.charAt(i);
         i++;
         matriz_temp += matriz.charAt(i);
       }
     else {
         matriz_temp += matriz.charAt(i);
       }
    }
    j = matriz_temp.length();


    int8_t tempIncidenceMatrix[j];

    k = 0;
    cont = 0;


    for (int  i = 0; i < j ; i++){
     temp = matriz_temp.charAt(i);
     if (temp == 45 ){
       tempIncidenceMatrix[k] = (temp);
       temp = matriz_temp.charAt(i+1);
       i++;
       tempIncidenceMatrix[k] += (temp-48);
       k++;
```

```
      cont++;
    }
    else {
      tempIncidenceMatrix[k] = (temp-48);
      k++;
    }
  }

  l = j- cont;

  int8_t IncidenceMatrix[l];


  for (int  i = 0; i < l ; i++){
   if (tempIncidenceMatrix[i] == 46){
     IncidenceMatrix[i] = (tempIncidenceMatrix[i]-47);
   }
   else {
     IncidenceMatrix[i] = (tempIncidenceMatrix[i]);
   }
  }
  Serial.print("Matriz de incid ncia ={");
  for (int  i = 0; i < l ; i++){
   Serial.print(IncidenceMatrix[i]);
   Serial.print(",");
  }
  Serial.println("}");



//Creation of the reader and PNRD objects
    Pn532NfcReader* reader = new Pn532NfcReader(&nfc);
    Pnrd pnrd = Pnrd(reader,int_places,int_transitions);//leitor, no esta

//Initialization of the communication with the reader and with the compu
    reader->initialize();

    pnrd.setIncidenceMatrix(IncidenceMatrix);
    pnrd.setTokenVector(StartingTokenVector);
```

```
//Setting of the classic PNRD approach
     pnrd.setAsTagInformation(PetriNetInformation::TOKEN_VECTOR);
     pnrd.setAsTagInformation(PetriNetInformation::ADJACENCY_LIST);
     Serial.print("\nMachine 1: Initial tag recording.");


}

void loop() {
  client_cloud.loop();
  delay(10);

  // check if connected
  if (!client_cloud.connected()) {
    connect();
  }

  // publish a message roughly every second.
  if (millis() - lastMillis > 300000) {
    lastMillis = millis();
    client_cloud.publish("PALMS-MQTT", "Arduino conectado");
  }



   Serial.print("Places = " );
   Serial.println(int_places);
   Serial.print("Transitions = " );
   Serial.println(int_transitions);


///Saving PNRD values in Tag////
  Serial.println("Place an tag to be configurated.");


  if(pnrd.saveData() == WriteError::NO_ERROR){
        Serial.println("Tag configurated successfully.");
  };
```

```
  Serial.print('\n');
  delay(1000);

}


void connect() {
    Serial.print("Conectando...");
    while (!client_cloud.connect("Arduino", "mqtteste", "testes")){
        Serial.print(".");
        delay("1000");
    }
    Serial.println("\nConectado!");

    client_cloud.subscribe("PALMS-MQTT");
}


void messageReceived(String &topic, String &payload){
  Serial.println(topic + ": " + payload);
  if (payload != "Fim"){
    total_msg += payload;
    total_msg += '\n';
  }
  if (payload == "Iniciar") {
     total_msg = "";
  }
  if (payload == "Fim") {
     Serial.println("Salvando no cart o SD");
     SD.remove("PNRDINFO.pnrd");
     myFile = SD.open("PNRDINFO.pnrd", FILE_WRITE);
     myFile.println(total_msg);
     myFile.close();
  }

  if (payload == "Info") {

  }
 }
```

D APÊNDICE - PROGRAMA DO ARDUINO® LEITOR

Programa do Arduino® Leitor no Arduinotextregistered Mega.

```
#include <MQTT.h>
#include <Ethernet.h>
#include <SPI.h>
#include "SdFat.h"
#include <Pn532NfcReader.h>
#include <PN532_HSU.h>
#include <PN532.h>
#include <PN532_SPI.h>
#include <NfcAdapter.h>



byte mac[] = {0xE4, 0x83, 0x26, 0xC4, 0x9a, 0xED};
byte ip[] = {10, 0, 208, 12};
byte gate[] = {10 , 0, 208, 1};
byte masc[] = {255, 255, 252, 0};
byte dns[] = {200, 19, 146, 201};


#define SD_CS_PIN 4
File myFile;
SdFat SD;

unsigned long lastMillis = 0;
String total_msg;

uint8_t int_places;
uint8_t int_transitions;

String places;
String transitions;
```

```
int contador;

String inf;


EthernetClient net;
MQTTClient client_cloud;

//Rotines related with the configuration of the RFID reader PN532 1
PN532_HSU pn532hsu1(Serial1);
NfcAdapter nfc1 = NfcAdapter(pn532hsu1);

//Rotines related with the configuration of the RFID reader PN532
2
PN532_HSU pn532hsu2(Serial2);
NfcAdapter nfc2 = NfcAdapter(pn532hsu2);

//Creation of the reader and PNRD objects 1
Pn532NfcReader* reader1 = new Pn532NfcReader(&nfc1);
Pnrd pnrd1 = Pnrd(reader1,int_places,int_transitions);//leitor, no estado

//Creation of the reader and PNRD objects  2
Pn532NfcReader* reader2 = new Pn532NfcReader(&nfc2);
Pnrd pnrd2 = Pnrd(reader2,int_places,int_transitions);//leitor, no estado

uint32_t tagId1 = 0xFF;
uint32_t tagId2 = 0xFF;


bool tagReadyToContinue = false;


void setup() {

    Serial.begin(115200);

  // Open serial communications and wait for port to open:
  while (!Serial) {
```

```
    ; // wait for serial port to connect. Needed for native USB port only
  }


  Serial.print("Initializing␣SD␣card...");

  if (!SD.begin(SD_CS_PIN)) {
    Serial.println("initialization␣failed!");
    return;
  }
  Serial.println("initialization␣done.");


// Configura  es para o uso do MQTT
    Ethernet.begin(mac, ip, dns, gate, masc);
    client_cloud.begin("mqtteste.cloud.shiftr.io", net);
    client_cloud.onMessage(messageReceived);
    connect();



//////LENDO CARTO SD///////

    Serial.println("Lendo␣o␣cart o␣SD");
    contador = 0;
    myFile = SD.open("PNRDINFO.pnrd");
    if (myFile) {
      while (myFile.available()) {
        String list = myFile.readStringUntil('\n');
        contador++;
        if (contador == 1){
          places = list;
          Serial.println("Places␣=␣" + places);
       }
       if (contador == 2){
          transitions = list;
          Serial.println("Transitions␣=␣" + transitions);
       }
       }
```

```
    // close the file:
    myFile.close();
  } else {
    // if the file didn't open, print an error:
    Serial.println("error opening test.txt");
  }



/////DESCOBRINDO VALORES DE PLACE E TRANSITIONS//////

  int_places = places.toInt();
  int_transitions = transitions.toInt();
  Serial.print("Places = " );
  Serial.println(int_places);
  Serial.print("Transitions = " );
  Serial.println(int_transitions);


//Creation of the reader and PNRD objects 1
    Pn532NfcReader* reader1 = new Pn532NfcReader(&nfc1);
    Pnrd pnrd1 = Pnrd(reader1,int_places,int_transitions);//leitor, no es

//Initialization of the communication with the reader and with the compu
    reader1->initialize();


//Setting of the classic PNRD approach
  pnrd1.setAsTagInformation(PetriNetInformation::TOKEN_VECTOR);
  pnrd1.setAsTagInformation(PetriNetInformation::ADJACENCY_LIST);



//Creation of the reader and PNRD objects
    Pn532NfcReader* reader2 = new Pn532NfcReader(&nfc2);
    Pnrd pnrd2 = Pnrd(reader2,int_places,int_transitions);//leitor, no es

//Initialization of the communication with the reader and with the compu
    reader2->initialize();
```

```
//Setting of the classic PNRD approach
  pnrd2.setAsTagInformation(PetriNetInformation::TOKEN_VECTOR);
  pnrd2.setAsTagInformation(PetriNetInformation::ADJACENCY_LIST);

  Serial.println("\nMachine 2: Reader.");

}

void loop() {
  client_cloud.loop();
  delay(10);

  // check if connected
  if (!client_cloud.connected()) {
    connect();
  }

  // publish a message roughly every second.
  if (millis() - lastMillis > 300000) {
    lastMillis = millis();
    client_cloud.publish("PALMS-MQTT", "Arduino conectado");
  }



  delay(1000);

  //Tag reading
  ReadError readError = pnrd1.getData();

 //In case of a successful reading
  if(readError == ReadError::NO_ERROR){

    FireError fireError;

    //Checks if it's a new tag
    if(tagId1 != pnrd1.getTagId()){
```

```
        tagId1 = pnrd1.getTagId();
        Serial.print("\nNew part. ID: ");
        Serial.println(tagId1, HEX);

        tagReadyToContinue = false;

         //Fire of transition 0
        FireError fireError = pnrd1.fire(0);

    switch(fireError){
        case FireError::NO_ERROR :

          //Save new information in tag
          if(pnrd1.saveData() == WriteError::NO_ERROR){
                Serial.println("NO ERROR.");
                inf = "NO ERROR.";
                return;
          }else{
                Serial.println("Error in the tag.");
                return;
          };
          return;

        case FireError::PRODUCE_EXCEPTION :
          Serial.println("Error: exception.");
          inf = "PRODUCE EXCEPTION.";
          break;

        case FireError::CONDITIONS_ARE_NOT_APPLIED :
          Serial.println("Error: CONDITIONS ARE NOT APPLIED");
          inf = "CONDITIONS ARE NOT APPLIED";
          break;
    }



    }
}
```

```
  delay(1000);

  //Tag reading
  ReadError readError2 = pnrd2.getData();

 //In case of a successful reading
  if(readError2 == ReadError::NO_ERROR){

    FireError fireError;

    //Checks if it's a new tag
    if(tagId2 != pnrd2.getTagId()){
        tagId2 = pnrd2.getTagId();
        Serial.print("\nNew part. ID: ");
        Serial.println(tagId2, HEX);

        tagReadyToContinue = false;

         //Fire of transition 0
        FireError fireError = pnrd2.fire(0);

    switch(fireError){
        case FireError::NO_ERROR :

          //Save new information in tag
          if(pnrd1.saveData() == WriteError::NO_ERROR){
                Serial.println("NO ERROR.");
                inf = "NO ERROR.";
                return;
          }else{
                Serial.println("Error in the tag.");
                return;
          };
          return;

        case FireError::PRODUCE_EXCEPTION :
```

```
            Serial.println("Error: exception.");
            inf = "PRODUCE EXCEPTION.";
            break;

          case FireError::CONDITIONS_ARE_NOT_APPLIED :
            Serial.println("Error: CONDITIONS ARE NOT APPLIED");
            inf = "CONDITIONS ARE NOT APPLIED";
            break;
      }




    }
  }

}


void connect() {
    Serial.print("Conectando...");
    while (!client_cloud.connect("Arduino", "mqtteste", "testes")){
        Serial.print(".");
        delay("1000");
    }
    Serial.println("\nConectado!");

    client_cloud.subscribe("PALMS-MQTT");
}


void messageReceived(String &topic, String &payload){
  Serial.println(topic + ": " + payload);
  if (payload != "End"){
    total_msg += payload;
    total_msg += '\n';
  }
  if (payload == "Atualizar") {
     total_msg = "";
  }
```

```
  if (payload == "End") {
      Serial.println("Salvando no cartao SD");
      SD.remove("PNRDINFO.pnrd");
      myFile = SD.open("PNRDINFO.pnrd", FILE_WRITE);
      myFile.println(total_msg);
      myFile.close();
  }

  if (payload == "Info") {
     client_cloud.publish("PALMS-MQTT", inf);

  }
}
```