UNIVERSIDADE FEDERAL DE GOIÁS

ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO

CURSO DE ENGENHARIA MECÂNICA

MANUFATURA DIGITAL: UMA CONTRIBUIÇÃO DA TECNOLOGIA PNRD APLICADA AO CONTROLE DE INVENTÁRIOS

POLLYANA ALVES RESENDE

GOIÂNIA – GO

2019

sistema de bibliotecas ufg

PRPG
PRÓ-REITORIA DE PÓS-GRADUAÇÃO

UFG

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TESES E DISSERTAÇÕES NA BIBLIOTECA DIGITAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFG), regulamentada pela Resolução CEPEC nº 832/2007, sem ressarcimento dos direitos autorais, de acordo com a Lei nº 9610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou *download*, a título de divulgação da produção científica brasileira, a partir desta data.

1. Identificação do material bibliográfico: [ ] Dissertação [ ] Tese [x] Monografia

2. Identificação da Tese ou Dissertação:

Nome completo do autor: Pollyana Alves Resende

Título do trabalho: **MANUFATURA DIGITAL: UMA CONTRIBUIÇÃO DA TECNOLOGIA PNRD APLICADA AO CONTROLE DE INVENTÁRIOS**

3. Informações de acesso ao documento:

Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

Havendo concordância com a disponibilização eletrônica, torna-se imprescindível o envio do(s) arquivo(s) em formato digital PDF da tese ou dissertação.

Pollyana Alves Resende
Assinatura do(a) autor(a)[2]

Ciente e de acordo:

João Paulo da Silva Fonseca
Assinatura do(a) orientador(a)[2]

Data: 19 / 07 / 2019

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. A extensão deste prazo suscita justificativa junto à coordenação do curso. Os dados do documento não serão disponibilizados durante o período de embargo.
Casos de embargo:
- Solicitação de registro de patente;
- Submissão de artigo em revista científica;
- Publicação como capítulo de livro;
- Publicação da dissertação/tese em livro.

[2] A assinatura deve ser escaneada.

POLLYANA ALVES RESENDE

MANUFATURA DIGITAL: UMA CONTRIBUIÇÃO DA TECNOLOGIA PNRD APLICADA AO CONTROLE DE INVENTÁRIOS

Monografia de conclusão de curso apresentada como requisito parcial para obtenção do título de Bacharel em Engenharia Mecânica pela Universidade Federal de Goiás.

Orientador:

Prof. Dr. João Paulo da Silva Fonseca

GOIÂNIA – GO

2019

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Resende, Pollyana Alves
Manufatura Digital: Uma Contribuição da Tecnologia PNRD Aplicada ao Controle de Inventários [manuscrito] / Pollyana Alves Resende. - 2019.
72 f.: il.

Orientador: Prof. Dr. João Paulo da Silva Fonseca.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Goiás, Escola de Engenharia Elétrica, Mecânica e de Computação (EMC), Engenharia Mecânica, Cidade de Goiás, 2019.
Bibliografia. Apêndice.
Inclui lista de figuras, lista de tabelas.

1. Indústria 4.0. 2. RFID. 3. PNRD. 4. Arduino. 5. Interface de aplicação. I. Fonseca, João Paulo da Silva, orient. II. Título.

CDU 621

POLLYANA ALVES RESENDE

MANUFATURA DIGITAL: UMA CONTRIBUIÇÃO DA TECNOLOGIA PNRD APLICADA AO CONTROLE DE INVENTÁRIOS

Projeto Final de curso, apresentado à Universidade Federal de Goiás, como parte das exigências para a obtenção do título de Bacharel em Engenharia Mecânica.

Goiânia, 16 de julho de 2019

BANCA EXAMINADORA

Prof. Dr. João Paulo da Silva Fonseca
Universidade Federal de Goiás

Prof. Dr. Sigeo Kitatani Júnior
Universidade Federal de Goiás

Prof. Dr. Marco Antônio Assfalk de Oliveira
Universidade Federal de Goiás

Prof. Me. Marco Vinícius Muniz Ferreira
Faculdade UNA

Aos meus pais, irmãos, amigos e colegas, pelo incentivo e pelo apoio constante.

AGRADECIMENTOS

Toda a minha gratidão

À Universidade Federal de Goiás e todo seu corpo docente, além da direção e a administração, que realizam seu trabalho com tanto amor e dedicação, trabalhando incansavelmente para que nós, alunos, possamos contar com um ensino de extrema qualidade.

Ao meu orientador, João Paulo da Silva Fonseca, pelo voto de confiança, pelo apoio constante, pela paciência, dedicação e ensinamentos que possibilitaram que eu realizasse este trabalho.

Ao Prof. Assfalk, por disponibilizar um tempo para ajudar no projeto.

Ao técnico Anderson, por acompanhar as atividades realizadas no Laboratório de Mecânica Aplicada (LabMEC).

Ao meu amigo Júlio Esteves, pela companhia no laboratório e pelos momentos de descontração.

Ao meu primo e amigo Vítor Nunes, pelo apoio técnico no projeto e por compartilhar bons momentos em família.

Aos meus amigos Eduardo Gonçalves, Henrique Gebran e Daniel Newman, por estarem comigo todos os dias desde o início da graduação nos momentos tristes e alegres, por todos os trabalhos em grupo, pelas viagens, pelos jogos juntos e pelas caminhadas até Trindade.

À toda minha família, especialmente minha mãe que sempre me incentivou desde criança a seguir com os estudos, e não mediu esforços para que isso acontecesse.

Aos meus demais amigos, por confiarem em mim e estarem do meu lado em todos os momentos da vida.

À empresa CSA (Centro de Serviços Aeronáuticos Ltda.), pela oportunidade de estágio e pela parceria organizacional, contribuindo para o meu crescimento profissional.

RESUMO

RESENDE, P.A. **Manufatura Digital: Uma Contribuição da Tecnologia PNRD Aplicada ao Controle de Inventários.** 2019. 72f. Monografia (Graduação em Engenharia Mecânica) – Escola de Engenharia Elétrica, Mecânica e de Computação, Universidade Federal de Goiás, Goiânia, 2019.

A Indústria 4.0 é um tema muito discutido no setor industrial atualmente, caracterizando-se por um conjunto de tecnologias que possuem o intuito de tornar os processos cada vez mais autônomos e eficientes. Dentro deste conceito estão inclusos a rastreabilidade de ativos e o controle de inventários, e a tecnologia de Identificação por Radiofrequência (RFID) é o sistema mais utilizado nessa aplicação. Recentemente, uma abordagem inovadora denominada Redes de Petri inseridas em base de dados RFID (PNRD) e sua variação, a PNRD invertida, foram desenvolvidas por pesquisadores da área de automação. Essa abordagem utiliza o modelo matemático desenvolvido das redes de Petri para definir uma estrutura de dados formal a ser inserida em etiquetas RFID, resultando em um sistema de controle com alta aplicabilidade. Nesse contexto, este trabalho tem como objetivo desenvolver uma interface de aplicação que se integre ao sistema RFID e a um banco de dados, possibilitando o controle de ativos de uma empresa. Dessa forma, descreve-se a modelagem do sistema, a aplicação da tecnologia PNRD embarcada em Arduino e a integração das funcionalidades do sistema. Os principais resultados estão voltados para a interface desenvolvida e como esta se integra à PNRD. O protótipo desenvolvido mostrou-se aplicável, fortalecendo a integração entre usuário da interface, ativos monitorados e elementos RFID. Por fim, vantagens e limitações do sistema desenvolvido são apresentadas, assim como a indicação de melhorias necessárias para aplicação em ambiente industrial.

Palavras-chave: Indústria 4.0, RFID, PNRD, Arduino, interface de aplicação.

ABSTRACT

RESENDE, P.A. **Manufatura Digital: Uma Contribuição da Tecnologia PNRD Aplicada ao Controle de Inventários.** 2019. 72f. Monografia (Graduação em Engenharia Mecânica) – Escola de Engenharia Elétrica, Mecânica e de Computação, Universidade Federal de Goiás, Goiânia, 2019.

Industry 4.0 is a much discussed topic in the industrial sector today, characterized by a set of technologies that aim to make processes more autonomous and efficient. Included within this concept are asset traceability and inventory control, and Radio Frequency Identification (RFID) technology is the most widely used system in this application. Recently, an innovative approach called Petri Nets inside RFID database (PNRD) and its variation, the inverted PNRD, were developed by automation researchers. This approach uses the developed mathematical model of Petri nets to define a formal data structure to be inserted into RFID tags, resulting in a control system with high applicability. In this context, this work aims to develop an application interface that integrates the RFID system and a database, allowing the control of the assets of a company. In this way, the system modeling, the application of the PNRD technology embedded in Arduino and the integration of the system's functionalities are described. The main results are focused on the developed interface and how it integrates with the PNRD. The developed prototype proved to be applicable, strengthening the integration between user interface, monitored assets and RFID elements. Finally, advantages and limitations of the developed system are presented, as well as the indication of the necessary improvements for application in an industrial environment.

Key-words: Industry 4.0, RFID, PNRD, Arduino, application interface.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O conceito de Indústria 4.0 é um tema muito recorrente e que vem sendo implantado com o intuito de tornar os processos cada vez mais eficientes, autônomos e customizáveis. A quarta revolução industrial se caracteriza por um conjunto de tecnologias que permitem a fusão do mundo físico, digital e biológico (ABDI, 2019). As principais tecnologias dentro desse contexto são: a Manufatura Aditiva, a Inteligência Artificial (IA), a Internet das Coisas, ou *Internet of Things* (IoT), a Biologia Sintética e os Sistemas Ciber-Físicos, ou *Cyber-Physical Systems* (CPS).

Segundo Silveira (2016), existem seis princípios para o desenvolvimento e implantação da Indústria 4.0: capacidade de operação em tempo real, virtualização, descentralização, orientação a serviços, modularidade e interoperabilidade. Aliados a esses princípios estão os avanços tecnológicos da última década nas áreas de tecnologia da informação e engenharia, como a Internet das Coisas, o *Big Data Analytics* e a segurança dos sistemas de informação.

No contexto dessas tecnologias, a rastreabilidade de ativos e o controle de inventários destacam-se como pontos chave dessa revolução. Para tal, a tecnologia de Identificação por Rádio Frequência, ou *Radio Frequency Identification* (RFID) é o sistema mais utilizado atualmente, uma vez que se mostra mais eficiente do que os códigos de barra e as faixas magnéticas. Essa tecnologia, apesar de existir desde a segunda guerra mundial (Domdouzi et al., 2007, apud Silva, 2017), tornou-se mais atrativa comercialmente nas últimas duas décadas, estimulando a troca dos tipos de identificação antigos pelo RFID.

O sistema RFID baseia-se na comunicação entre um leitor e uma (ou várias) etiqueta, através de uma antena. Essa comunicação é dada por meio de campos eletromagnéticos que permitem a leitura e gravação de dados nas etiquetas, ou *tags*. Esses dispositivos podem ser fixados em pessoas, animais ou objetos a fim de acompanhar sua movimentação e obter os seus dados naquele instante.

A tecnologia RFID é amplamente utilizada em aplicações como no controle de acesso a locais, em sistemas de transporte público, na identificação de animais, em veículos para recolhimento de pedágio/estacionamento automático, no controle de instrumentos cirúrgicos em hospitais, no controle de acervos em bibliotecas e, principalmente, na rastreabilidade industrial, identificando produtos manufaturados ao longo da cadeia de produção, distribuição, etc.

Para unir o sistema de identificação com o controle de movimentações de ativos, uma ferramenta de análise de sistemas discretos deve ser utilizada. A rede de Petri é uma técnica de

modelagem que permite a representação de sistemas, utilizando como alicerce uma forte base matemática (Maciel et al., 1996, apud Francês, 2003). Essa técnica permite modelar o fluxo de trabalho dos ativos, identificando quais os possíveis caminhos por onde este pode passar, qual a sequência desses locais e quais as condições para que possa passar em cada local. Assim, caso o objeto passe por um local indesejado ou no tempo incorreto, o sistema é capaz de detectar falhas e ações corretivas devem ser tomadas.

Seguindo essa linha, Tavares e Saraiva (2010) propuseram um método de integração do modelo de redes de Petri com sistemas de controle RFID, denominado de Rede de Petri inserida em base de dados RFID, ou *Petri Net inside RFID database* (PNRD) (apud Silva, 2017). Dessa forma, as etiquetas contêm as informações sobre o estado atual do ativo, o que pode ser levado para um banco de dados, atualizando a localização do ativo em tempo real.

Recentemente, Silva (2017) desenvolveu uma biblioteca para aplicações de PNRD embarcadas em Arduino, uma plataforma eletrônica de código aberto baseada em *hardware* e *software* fáceis de usar (Arduino, 2019). As placas Arduino são compostas por um microcontrolador e podem ser facilmente conectadas à um computador via USB e programadas via IDE (*Integrated Development Environment*, ou Ambiente de Desenvolvimento Integrado) utilizando uma linguagem baseada em C/C++ (Thomsen, 2014).

Para uma interação com o usuário, o Arduino IDE possui um monitor serial, em que valores impressos na porta serial podem ser visualizados e o usuário também pode entrar com dados para fazer escolhas. Porém, para que o sistema tenha uma interação mais fácil com o usuário, uma interface de controle é necessária. Para tal, um servidor *web* local é um dos programas mais utilizados. Sua programação geralmente é feita em linguagem HTML (*HyperText Markup Language*, ou Linguagem de Marcação de HiperTexto) e/ou PHP (um acrônimo recursivo para PHP: *Hypertext Preprocessor*, ou Pré-processador de Hipertexto).

Outro problema em utilizar apenas o monitor serial do Arduino IDE é que os dados lidos das etiquetas e os dados inseridos pelo usuário não são gravados, pois o Arduino não apresenta funções para criação, escrita e leitura de arquivos texto, sem que seja utilizado o módulo leitor de cartão SD. Assim, para suprir essa finalidade, sem a utilização de um componente extra, um sistema com banco de dados é uma possível solução. O MySQL é um sistema gerenciador de banco de dados relacional de código aberto que utiliza a linguagem SQL (*Structure Query Language*, ou Linguagem de Consulta Estruturada) para inserir, acessar e gerenciar o conteúdo armazenado no banco de dados (Pisa, 2012).

Para estabelecer a comunicação entre o Arduino e o banco de dados/PHP, um outro fator é necessário, uma vez que o Arduino não possui bibliotecas para interação com essas aplicações

de forma direta. Em vista disso, a primeira forma é utilizando um *Shield Ethernet* e uma biblioteca disponível para o Arduino que faz a conexão com o MySQL, assim a conexão se dá via *Ethernet*. A outra forma é através de um *script* Python, que disponibiliza bibliotecas tanto para comunicação com o Arduino, quanto para comunicação com o MySQL. O Python é uma linguagem dinâmica, interpretada, robusta, multiplataforma e multi-paradigma (orientação à objetos, funcional, refletiva e imperativa) (Lage, 2010). Dessa maneira, os dados são transmitidos pela porta Serial.

Portanto, observa-se a possibilidade de aplicação da biblioteca PNRD desenvolvida por Silva (2017) na implementação de uma interface para monitoramento de ativos, aliada a um banco de dados contendo as informações advindas do sistema RFID.

1.1 Objetivo Geral

O objetivo geral deste trabalho consiste em desenvolver uma interface de controle das ferramentas de uma empresa, a CSA – Centro de Serviços Aeronáuticos Ltda, aplicando diretamente uma técnica computacional em dispositivos de campo, voltada a tarefas de controle de inventários e ciclo de vida de produtos.

1.2 Objetivos Específicos

- Modelar o comportamento dos objetos de estudo em uma Rede de Petri;
- Identificar os objetos com o sistema RFID;
- Aplicar a tecnologia PNRD embarcada em Arduino;
- Armazenar informações em um banco de dados MySQL;
- Desenvolver uma interface de comunicação com o usuário, em PHP e HTML;
- Integrar as funcionalidades do sistema;
- Executar testes do protótipo para validação da proposta.

1.3 Estrutura

No capítulo 2 será abordada a fundamentação teórica deste trabalho, seguido da metodologia de desenvolvimento no capítulo 3, dos resultados no capítulo 4 e, por fim, das conclusões no capítulo 5.

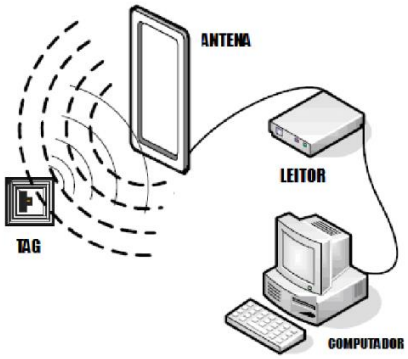
2 FUNDAMENTAÇÃO TEÓRICA

Como o desenvolvimento deste trabalho aborda a integração de diferentes assuntos, alguns conceitos essenciais serão melhor detalhados para maior entendimento do projeto.

2.1 Sistema RFID

A tecnologia RFID utiliza ondas eletromagnéticas entre 125 kHz e 5,8 GHz, com o intuito de armazenar e ler informações de dispositivos denominados etiquetas ou *tags* (Silva, 2017). Um sistema RFID é composto por três componentes físicos: etiqueta, antena e leitor. Ahsan, Shah e Kingston (2010, apud Tavares et al., 2018) citam outros dois componentes, responsáveis por controlar e manipular as informações contidas nas etiquetas: a infraestrutura de comunicação e o *software* de aplicação. A Figura 1 representa a interação entre esses componentes.

Figura 1 – Componentes de um sistema RFID



Fonte: Araújo, 2018.

O fluxo de dados para uma *tag* passiva geralmente ocorre da seguinte forma:

- Pelo computador, no *software* de aplicação, a leitura da etiqueta é solicitada;
- Essa solicitação é informada ao leitor através da infraestrutura de comunicação;
- O leitor emite um sinal de radiofrequência procurando por etiquetas próximas;
- A antena é responsável por transmitir esse sinal;
- Ao receber o sinal, a etiqueta envia a resposta para a antena;
- O leitor recebe da antena o sinal modulado e decodifica os dados presentes;
- Através da infraestrutura de comunicação, o *software* recebe os dados para a aplicação.

Existem vários tipos de leitores, antenas e etiquetas, que operam com frequências diferentes. Alguns desses modelos e uma maior descrição desses componentes serão apresentados a seguir.

2.1.1 Leitores

O leitor é o equipamento responsável por emitir e interpretar o sinal recebido pela antena. Esse dispositivo pode ser classificado como monoestático, quando o leitor possui apenas uma antena para emitir e receber o sinal de radiofrequência da etiqueta, ou como biestático, quando possui uma antena para transmitir e outra antena para receber o sinal (RFID Journal, 2019).

Os leitores RFID podem ser utilizados de duas formas: fixos ou móveis. Na primeira, o leitor é fixado em alguma estrutura não móvel e as *tags* anexadas aos objetos são lidas ao passar próximo ao leitor. Já na segunda forma, o leitor é quem acompanha a unidade em movimento, e as *tags* são fixas em pontos estratégicos para informar a posição do leitor. Alguns exemplos de leitores estão apresentados na Figura 2.

Figura 2 – Leitores RFID

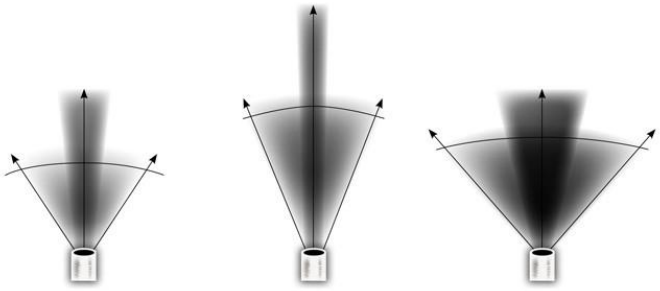
Fonte: Acura, 2019.

2.1.2 Antenas

A antena é o dispositivo responsável por estabelecer a comunicação entre o leitor e a etiqueta. Esse componente funciona como uma ponte, realizando a modulação dos sinais de radiofrequência enviados pelo leitor e a demodulação dos sinais recebidos das *tags* (Silva, 2017). As antenas possuem características construtivas importantes e podem ser classificadas quanto ao ganho de potência, ao alcance, à polarização, à diretividade e à frequência de operação.

O ganho de potência, expresso em dBi, refere-se à potência extra que antena pode oferecer ao sinal (Neto, 2015). Esse fator está diretamente relacionado à largura do feixe emitido pela antena, ou seja, quanto maior o ganho, maior a largura. A Figura 3 representa essa característica.

Figura 3 – Feixe de antenas direcionais de diferentes ganhos



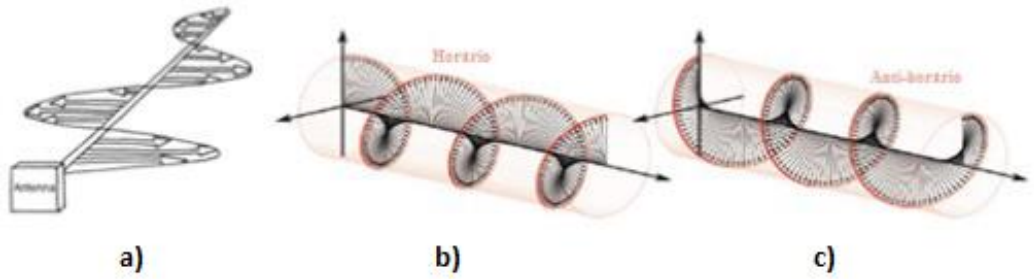
Fonte: Silva, 2017.

Quanto ao alcance, as antenas podem ser classificadas como de Campo Próximo, ou *Near Field*, ou de Campo Distante, ou *Far Field*. Se uma *tag* RFID está fora de um comprimento de onda completa do leitor, diz-se estar no "campo distante", e se estiver dentro, diz-se estar no "campo próximo". O sinal do campo distante decai o quadrado da distância da antena, enquanto o sinal de campo próximo decai o cubo da distância da antena. Então, sistemas passivos de RFID que dependem de comunicação de campo distante (tipicamente *Ultra High Frequency* e microondas) tem um alcance de leitura mais longo do que aqueles que usam comunicação de campo próximo (normalmente sistemas de baixa e alta frequência) (RFID Journal, 2019).

Quanto à polarização, as antenas podem ser classificadas como Linear ou Circular. As antenas lineares concentram a energia de rádio do leitor em uma orientação ou polaridade. A emissão se dá em apenas um plano, podendo ser horizontal ou vertical. Devido a isso, o sinal emitido é capaz de se propagar por maiores distâncias e com maior potência (Silva, 2017). Porém, as *tags* devem estar alinhadas ao leitor para que a leitura seja efetuada.

As antenas circulares emitem ondas de rádio de maneira helicoidal, podendo ser em sentido horário ou anti-horário (Silva, 2017). Essas antenas são usadas em situações em que a orientação da *tag* não pode ser controlada, pois como as ondas são propagadas em formato circular, as chances de o sinal emitido ser recebido são maiores. Porém, seu alcance de leitura é menor do que as antenas lineares (RFID Journal, 2019). A Figura 4 apresenta os tipos de polarização.

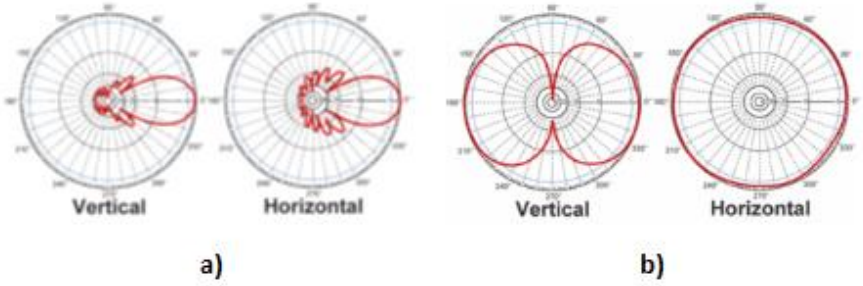
Figura 4 – Tipos de polarização: a) linear; b) circular – horário; c) circular – anti-horário



Fonte: Silva, 2017.

Quanto à diretividade, as antenas podem ser classificadas como Direcional ou Omnidirecional. As direcionais emitem o sinal apenas para a direção em que a antena está apontada. Já as omnidirecionais emitem o sinal para todas as direções (Silva, 2017). A Figura 5 apresenta esses dois tipos de diretividade.

Figura 5 – Tipos de diretividade: a) direcional; b) omnidirecional



Fonte: Silva, 2017.

As antenas operam em diversas frequências, podendo ser baixas, altas, ultra-altas e microondas (RFID Journal, 2019). Cada tipo de frequência tem suas vantagens e desvantagens, portanto é necessária uma análise de viabilidade para definir qual é a frequência mais adequada para se utilizar em certa aplicação. Vale ressaltar que todos os dispositivos do sistema devem operar na mesma frequência, para haver compatibilidade de sinais.

2.1.3 Etiquetas

A etiqueta é um microchip ligado a uma antena, embalado em uma forma que possa ser aplicado a um objeto (RFID Journal, 2019). Esse componente possui um número de série fixo e pode conter informações sobre o objeto que acompanha. Existem *tags* RFID de muitos formatos, como chaveiros, cartões e adesivos. Alguns tipos de etiquetas estão representados na Figura 6.

Figura 6 – Modelos de etiquetas RFID, da esquerda pra direita: chaveiro e adesivos

Fonte: própria autora.

As etiquetas podem ser classificadas como ativas, passivas ou semi-passivas. As ativas são caracterizadas por possuir um transmissor para enviar de volta informações, ao invés de só refletir o sinal do leitor, e geralmente possuem uma fonte de energia própria. Esse tipo de *tag* pode ser lido de grandes distâncias, porém são mais caras. As etiquetas passivas não possuem fonte de energia própria e nem um transmissor. As ondas de radiofrequência emitidas pelo leitor ativam o microchip da *tag*, que por sua vez envia seus dados para o leitor. Hoje em dia, essas etiquetas são bem acessíveis e custam em média US$ 0,20 (RFID Journal, 2019). As semi-passivas são semelhantes às ativas, porém a bateria é usada apenas para executar os circuitos do microchip e não para enviar o sinal para o leitor.

Algumas *tags* são do tipo regravável, ou *rewrite*, onde é possível modificar a informação salva por meio de ondas de radiofrequência, enquanto outras são do tipo apenas leitura, ou *read-only*, e para fazer essas alterações, é necessário reprogramar eletronicamente o microchip da etiqueta. Em alguns casos as etiquetas podem ser mistas, apresentando áreas de memória do tipo *read-only* e do tipo *rewrite* (Silva, 2017).

As etiquetas também podem ser classificadas pela frequência de operação, podendo ser baixas, altas, ultra-altas e super altas. A Tabela 1 apresenta essas faixas de frequências dos sistemas RFID.

Tabela 1 – Frequências de operação das etiquetas RFID

Tipo da Etiqueta	**Frequência**
Baixa frequência (LF)	125 – 134,3 kHz
Alta frequência (HF)	13,56 MHz
Ultra alta frequência (UHF)	860 – 960 MHz
Super alta frequência (SHF)	2,45 – 5,8 GHz

Fonte: Silva, 2017.

2.1.4 *Software* de aplicação

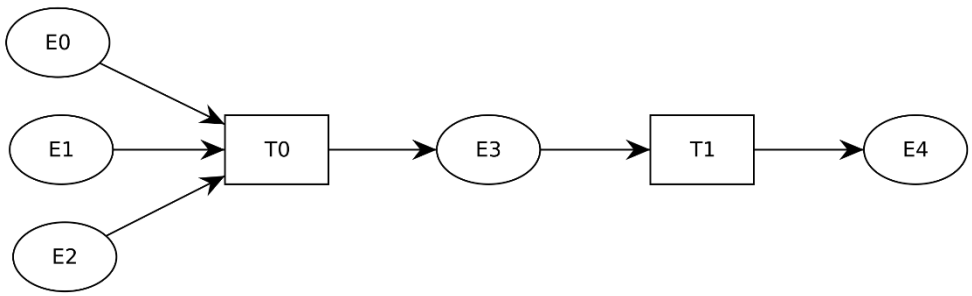
Através da infraestrutura de comunicação, os dados são transmitidos ao *software* de aplicação, que será responsável por processar a informação recebida. Geralmente, a aplicação está relacionada com um banco de dados, pois é interessante armazenar os dados das etiquetas em algum local para que possam ser usados em alguma aplicação posteriormente. Além disso, o *software* de aplicação pode conter uma interface de comunicação com o usuário, tornando possível visualizar, por exemplo, a localização de objetos, ou tomar ações que necessitem da interferência humana.

2.2 Redes de Petri

Segundo Murata (1989), redes de Petri consistem em um modelo conceitual gráfico e matemático introduzido por Carl Adam Petri em 1962. Para Jensen e Kristensen (2009), as redes de Petri fornecem a base da notação gráfica e das primitivas básicas para modelagem de concorrência, comunicação e sincronização (apud Tavares et al., 2018).

A representação gráfica das redes de Petri é composta por três componentes: os lugares, os arcos e as transições. Os lugares são representados por circunferências, as transições por traços ou retângulos e os arcos são setas que ligam os outros dois componentes, direcionando o fluxo da rede. Vale ressaltar que os arcos sempre ligam um lugar a uma transição, e nunca um lugar a um lugar ou uma transição a uma transição. A Figura 7 apresenta uma rede de Petri de cinco lugares e duas transições.

Figura 7 – Exemplo de rede de Petri



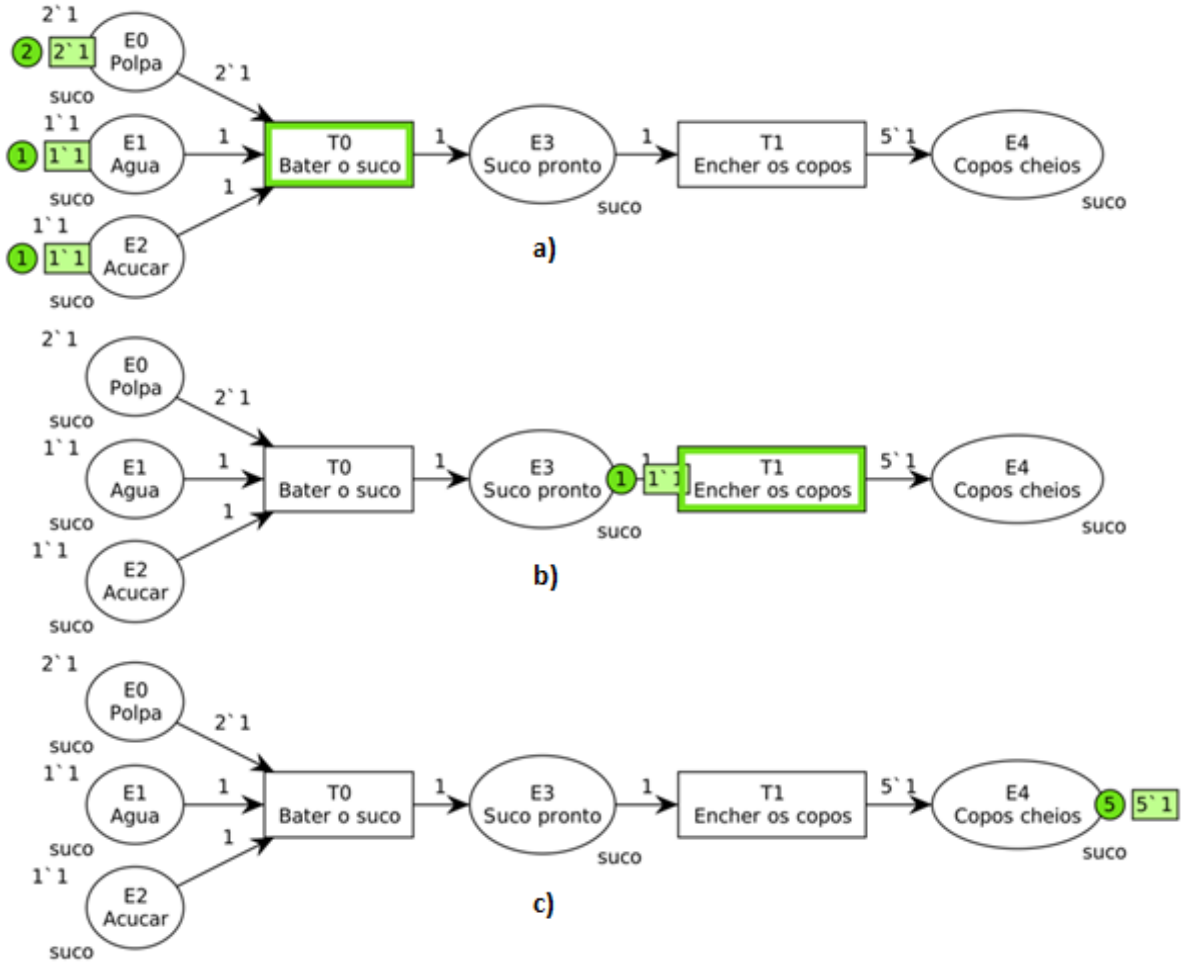
Fonte: própria autora.

Além desses três componentes, existem as marcações, ou *tokens*, que representam o estado em que se encontram os ativos, e são representadas por pontos. Por exemplo, tomando a rede de Petri da Figura 7 como a preparação de um suco, os lugares E0, E1 e E2 representam os insumos para produzir o suco, como a polpa, a água e o açúcar. A transição T0 é a ação

"bater o suco", e só será habilitada quando houver os três ingredientes necessários para isso. Depois de batido, o suco entra em um lugar E3 "pronto". Assim, a ação "encher os copos" na transição T1 é habilitada, e, por fim, o lugar E4 representa os copos servidos cheios de suco.

Para mudar de um lugar para outro, como, por exemplo, dos ingredientes para o suco pronto, um disparo na rede de Petri é efetuado. Um disparo acontece quando uma transição é acionada, mudando o *token* de seu lugar atual para o lugar em sequência. Uma transição só pode ser disparada quando o lugar anterior tiver marcações suficientes, dependendo da condição da transição. A Figura 8 representa as etapas da preparação do suco, com a quantidade de *tokens* necessários para cada disparo.

Figura 8 – Etapas da preparação do suco: a) etapa inicial; b) etapa intermediária; c) etapa final



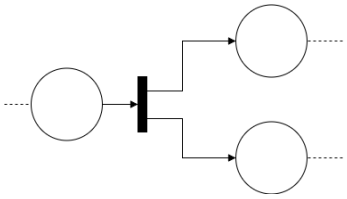
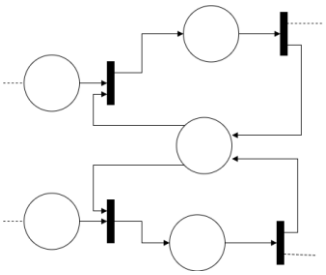
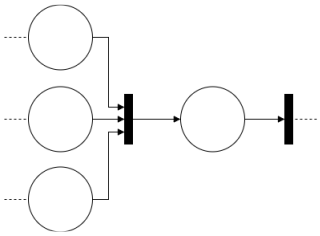
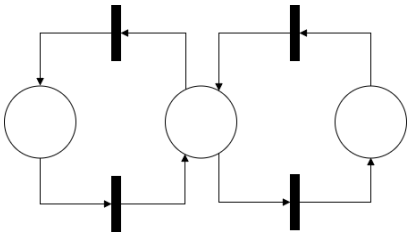
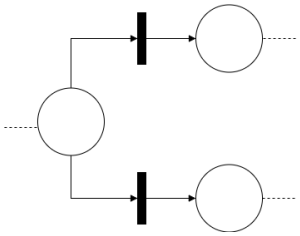
Fonte: própria autora.

Os arcos, entre os lugares e transições, podem possuir pesos. Nesses casos, é necessário que alguns lugares possuam mais *tokens* do que outros para que a transição seja disparada. Nota-se, na Figura 8a, que duas polpas são necessárias para apenas uma dose de água e uma de açúcar. Caso o arco esteja ligando uma transição a um lugar e ele tenha um peso dois, isso

significa que ao disparar essa transição, o próximo lugar ganhará dois *tokens* ao invés de apenas um. Nas Figuras 8b e 8c, ao passar do lugar E3 para o E4, o suco pronto se transforma em cinco copos cheios.

A grande vantagem da rede de Petri é a sua capacidade de representar sistemas caracterizados como sendo concorrentes, assíncronos, distribuídos, paralelos, não determinísticos e/ou estocásticos (Murata, 1989, apud Silva, 2017). A Tabela 2 apresenta essas possíveis representações da rede de Petri.

Tabela 2 – Características de sistemas não-sequenciais e a sua representação com redes de Petri

Característica do Sistema	**Possível representação**
Paralelismo	
Concorrência	
Rendez-vous	
Máquina de estados	
Escolha não-determinística	

Fonte: Silva, 2017.

2.2.1 Definição formal

De acordo com Silva (2017), a rede de Petri com marcações pode ser definida formalmente como sendo uma função de quatro elementos, *P*, *T*, *A*, *M*, sendo eles:

- *P* um conjunto finito não-nulo de lugares;
- *T* um conjunto finito não-nulo de transições;
- *A* um conjunto de arcos, tanto direcionados de lugares para transições quanto de transições para lugares, tal que $A \subseteq (P \times T) \cup (T \times P)$;
- *M* uma função do número de marcações presentes em cada lugar, de modo que $M: P \rightarrow IN$.

Um disparo pode ser caracterizado pela Equação 1 apresentada a seguir:

$$M_{k+1} = M_k + A^T * u_k \, , k = 0, 1, 2, \dots, n \qquad (1)$$

Onde:

- M_k é o vetor de marcações, definido como um vetor coluna cujo elemento da enésima linha corresponde a quantidade de marcações do enésimo lugar, em determinado instante de tempo *k*;
- M_{k+1} é o vetor de marcações resultante do disparo *k*;
- A^T é a matriz de incidência, que representa o conjunto de arcos da rede de Petri. O número de linhas é igual ao número de lugares da rede, e o número de colunas é igual ao número de transições. O elemento da matriz de incidência a_{ij} tem o valor de -1 quando o lugar *i* for uma entrada da transição *j* e tem o valor de 1 quando for uma saída. Caso o lugar não tenha nenhuma relação com a transição, o elemento terá valor 0.
- u_k é o vetor de disparos, definido como um vetor coluna cujo elemento da enésima linha corresponde a quantidade de vezes que a transição *n* foi realizada no disparo.

No exemplo do suco, visto nas Figuras 7 e 8, a matriz de incidência pode ser definida como:

$$A^T = \begin{bmatrix} -2 & 0 \\ -1 & 0 \\ -1 & 0 \\ 1 & -1 \\ 0 & 5 \end{bmatrix}$$

Os elementos da primeira coluna têm relação com a transição T0 e os da segunda coluna com a transição T1. Já as linhas estão relacionadas com os lugares, sendo a primeira linha o lugar E0 e as outras E1, E2, E3 e E4, respectivamente. Note que o lugar E0 possui valor -2 na transição T0, pois o arco que liga essas transições possui peso dois. Da mesma forma, o lugar E4 tem valor 5 na transição T1, uma vez que o arco possui peso cinco.

Considerando o conjunto de marcações iniciais, como mostrado na Figura 8, o vetor de marcações pode ser definido como:

$$M_0 = \begin{bmatrix} 2 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Ainda no instante inicial, a transição que está habilitada para ser disparada é a T0. Portanto, o vetor de disparos pode ser definido como:

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

Assim, aplicando a Equação 1, o disparo resulta no seguinte vetor de marcações:

$$M_1 = M_0 + A^T * u_0 = \begin{bmatrix} 2 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -2 & 0 \\ -1 & 0 \\ -1 & 0 \\ 1 & -1 \\ 0 & 5 \end{bmatrix} * \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -2 \\ -1 \\ -1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Esse resultado é válido e pode ser confirmado na Figura 8, na segunda imagem. Os três ingredientes (2 polpas, 1 água e 1 açúcar) após passarem pela transição T0 (bater o suco) chegaram como 1 no lugar E3 (suco pronto). Caso o vetor de marcações finais possua números negativos, o disparo resultou em uma exceção, significando que não havia marcações suficientes nos lugares de entrada para acionar as transições descritas no vetor de disparos (Silva, 2017).

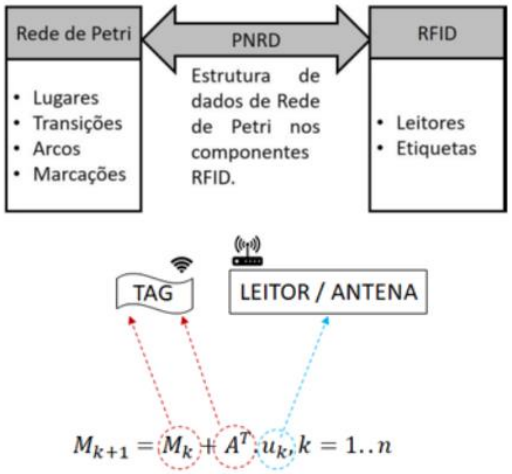
2.3 PNRD

A rede de Petri inserida em base de dados RFID, proposta por Tavares e Saraiva (2010), trata-se de uma estrutura de dados formal a ser armazenada em etiquetas RFID, para aplicações no nível operacional, como em sistemas de controle.

A PNRD foi concebida originalmente para processos logísticos e de manufatura, e pressupõe que o comportamento típico de um objeto possa ser modelado por meio de redes de Petri (Tavares et al., 2018). Para isso, o fluxo de trabalho da etiqueta relacionada ao objeto é definido pela matriz de incidência e seu vetor de marcações, indicando sua posição atual. Além desses dois dados, a ID da etiqueta e uma indicação de tempo, para possibilitar a análise quantitativa de performance, constituem a estrutura típica da PNRD.

Levando em conta a definição formal da rede de Petri, as etiquetas estão relacionadas à matriz de incidência e ao vetor de marcações atual, enquanto os leitores e as antenas estão associados ao vetor de disparos, para que quando haja uma leitura, o disparo seja efetuado e o vetor de marcações da *tag* seja atualizado. Essa estrutura pode ser visualizada na Figura 9.

Figura 9 – PNRD como ponte entre rede de Petri e RFID



Fonte: Tavares et al., 2018.

Dessa forma, o objeto etiquetado contem armazenado internamente o processo pelo qual deve passar. Assim, caso o disparo gere uma exceção, ou seja, o objeto passe por um caminho pelo qual não deveria, o próprio sistema é capaz de reconhecer e tratar o erro em tempo de execução.

Outra abordagem nesse campo de estudo é a PNRD invertida (iPNRD). Desenvolvida por Fonseca e Tavares (2017), no contexto de busca e salvamento de pessoas em trilhas de caminhada. Essa abordagem busca viabilizar a utilização do conceito da PNRD em aplicações cuja a instalação de antenas RFID nos pontos de transição seja demasiadamente difícil (Silva, 2017).

Ao contrário da PNRD convencional, esse tipo é caracterizado por ter as etiquetas em pontos fixos e os leitores em movimento. Dessa maneira, as *tags* contêm em sua memória

apenas o vetor de disparos, enquanto os leitores são responsáveis por armazenar o estado do sistema, definido pela matriz de incidência e pelo vetor de marcações.

Uma exceção nesse caso significa que o portador do leitor está seguindo o caminho errado, ou realizando uma ação que não deveria executar, informações diretamente associadas a pessoas que tenham se perdido ou se machucado durante uma trilha. Esse novo tipo também traz como diferencial o fato de que as etiquetas precisam armazenar um histórico de visitas de leitores, para que seja possível consultar posteriormente a rota seguida, permitindo direcionar possíveis ações de busca e salvamento para uma região mais provável de se encontrar a vítima.

2.4 Arduino e biblioteca PNRD

Arduino nasceu no Instituto de Design de Interação Ivrea (*Ivrea Interaction Design Institute*) como uma ferramenta fácil para prototipagem rápida, destinada a estudantes sem formação em eletrônica e programação (Arduino, 2019). Essa plataforma possui um microcontrolador que pode ser facilmente programado via USB e que é capaz de ler entradas, como por exemplo, sinais de sensores ou o acionamento de um botão, e transformá-los em saídas, como o acionamento de um LED ou a partida de um motor.

As plataformas Arduino diferenciam-se em alguns aspectos, como o microcontrolador, a capacidade de processamento, a quantidade memória e o número de portas digitais e analógicas. Por exemplo, o Arduino Uno possui um microcontrolador ATmega328, com um *clock* de 16 MHz, 32 KB de memória, 14 portas digitais e 6 portas analógicas. Por outro lado, o Arduino Mega 2560 possui um microcontrolador ATmega2560, com um *clock* de 16 MHz, 256 KB de memória, 54 portas digitais e 16 portas analógicas. A Figura 10 ilustra esses dois modelos de Arduino.

Figura 10 – Arduinos, Uno e Mega2560

Fonte: própria autora.

Algumas vantagens na utilização dessa plataforma estão relacionadas com: baixo custo de aquisição, plataforma *open-source*, compatibilidade da plataforma com diferentes sistemas, programação simples para iniciantes sem conhecimento na área, grande comunidade de desenvolvedores, *software* e *hardware* de código aberto e fáceis de usar e grande disponibilidade de bibliotecas sem custo adicional.

Utilizando-se dessas vantagens, Silva (2017) desenvolveu uma biblioteca para aplicações de PNRD e iPNRD para o Arduino. A biblioteca foi dividida em três módulos: a rede de Petri, a PNRD e o leitor de etiquetas. O primeiro é responsável pela base matemática da biblioteca, permitindo a definição da matriz de incidência, do vetor de marcações e do vetor de disparos, bem como o cálculo dos disparos e a verificação de exceções.

O módulo da PNRD é responsável por integrar o sistema RFID e o módulo da rede de Petri, permitindo aplicações tanto da PNRD convencional, quanto da invertida. Para diferenciar os dois tipos foram criadas rotinas de configuração, determinando quais dados são armazenados na etiqueta e quais são de responsabilidade do controlador.

Por fim, o terceiro módulo é o do leitor PN532, que contém os métodos de leitura e gravação nas etiquetas utilizando a antena de modelo PN532. Dessa forma, caso seja necessário trocar de leitor em certa aplicação, somente esse módulo precisará ser substituído, reaproveitando os demais.

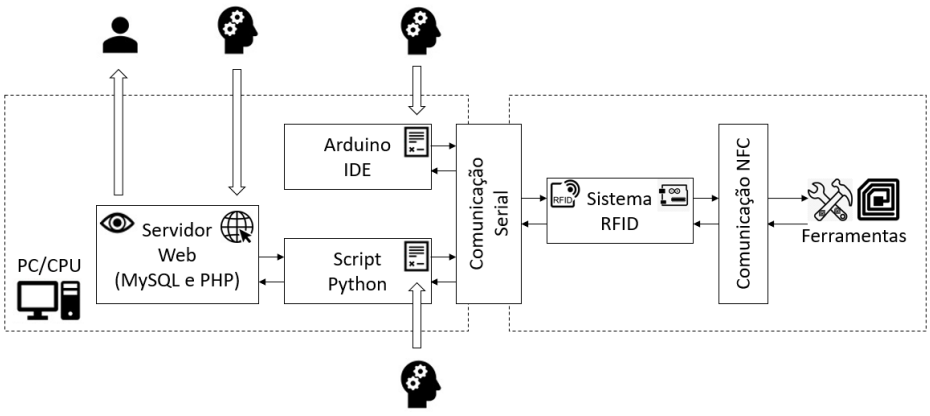
3 METODOLOGIA

Considerando o objetivo deste trabalho, e o fato de que a utilização de dois leitores foi sugerida, a metodologia para a aplicação da PNRD no controle de movimentações de ferramentas foi dividida em duas etapas. A primeira, relacionada ao Leitor 1, opera com comunicação serial, uma vez que o Arduino estará conectado diretamente ao computador que possuirá o *software*. Já a segunda etapa, relacionada ao Leitor 2, possui uma comunicação *Ethernet*, uma vez que o Arduino estará fisicamente distante do computador no qual estará instalado o servidor.

Antes de efetuar essa divisão em duas etapas, primeiramente o objeto de estudo foi definido e seu comportamento foi modelado em uma Rede de Petri, determinando os possíveis caminhos por onde a ferramenta pode se movimentar. Para acompanhar esse processo, o sistema RFID foi utilizado, de modo que a posição em tempo real da ferramenta seja conhecida. O fluxo de dados na parte física é dado entre etiqueta e leitor RFID, através da comunicação por campo próximo, ou *Near Field Communication* (NFC). Já no ambiente computacional, entre o sistema RFID e o Arduino IDE, os dados trafegam via comunicação Serial.

Para o Leitor 1, a informação advinda do sistema RFID é gravada em um banco de dados MySQL, através de um *script* Python. Por fim, para a visualização dessas movimentações e para o cadastro de informações que não acompanham a etiqueta, como por exemplo, a ordem de serviço e o nome do técnico responsável, uma API (*Application Programming Interface*) foi desenvolvida utilizando um servidor *web*, programado em PHP e HTML. Um esquema ilustrativo do modo de operação do sistema RFID para o Leitor 1, baseado em Fonseca (2018), é ilustrado na Figura 11.

Figura 11 – Esquema ilustrativo do modo de operação do sistema RFID para o Leitor 1

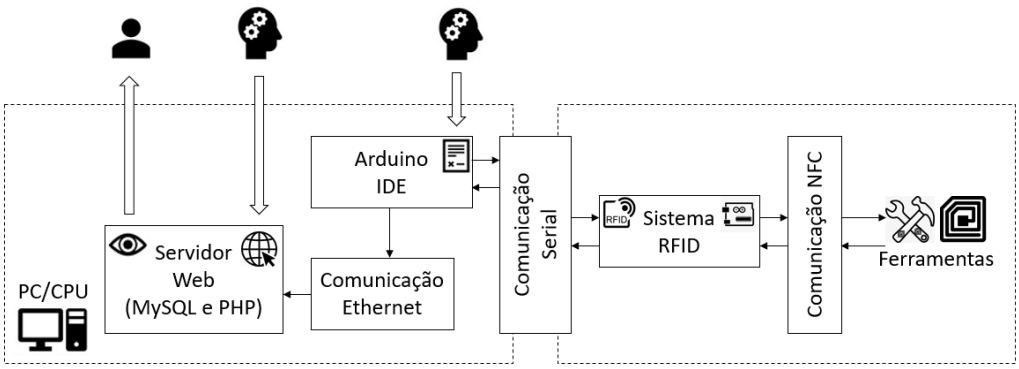


Fonte: própria autora, modificado de Fonseca (2018).

Nota-se a necessidade de desenvolvimento de lógica computacional em três processos distintos: o código computacional embarcado na plataforma Arduino, o desenvolvimento do servidor *web* e o *script* Python para possibilitar a comunicação entre os dois primeiros. Os dados nesse ambiente computacional trafegam apenas via comunicação serial.

Para o Leitor 2, a informação advinda do sistema RFID é gravada no mesmo banco de dados MySQL, através da comunicação *Ethernet*. Através de uma biblioteca disponível para o Arduino, essa comunicação pode ser estabelecida, desde que um novo *hardware*, denominado *Shield Ethernet*, seja acrescentado a esta aplicação. Nesse caso, os dados trafegam em apenas um sentido, do Arduino IDE para o servidor *web*, pois a movimentação será solicitada no servidor pelo Leitor 1, sendo necessária somente sua confirmação. Assim, os dados das etiquetas que passam por esse leitor podem ser visualizados na API. Um esquema ilustrativo do modo de operação do sistema RFID para o Leitor 2, baseado em Fonseca (2018), é ilustrado na Figura 12.

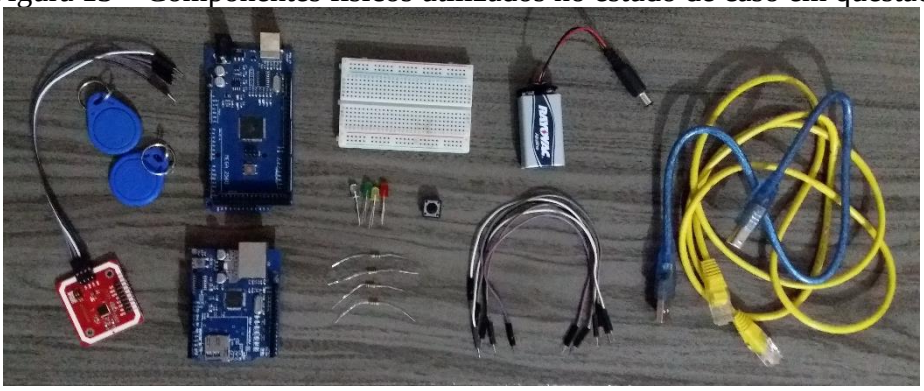
Figura 12 – Esquema ilustrativo do modo de operação do sistema RFID para o Leitor 2



Fonte: própria autora, modificado de Fonseca (2018).

O protótipo desenvolvido utiliza um computador onde executa o servidor, dois Arduinos Mega2560, dois leitores RFID NFC PN532, etiquetas RFID Mifare de 1 KB do tipo chaveiro, compatíveis com a frequência de operação do leitor (13,56 MHz), um *Shield Ethernet*, um cabo *Ethernet*, um cabo USB, uma bateria de 9V, oito resistores de 100 Ω, um botão de pulso, sete LED's, dois *protoboards* de 400 pontos e jumpers macho-macho e macho-femêa para a ligação do circuito. Esses componentes, não em suas quantidades totais, são apresentados na Figura 13.

Figura 13 – Componentes físicos utilizados no estudo de caso em questão



Fonte: própria autora.

Já na parte computacional, vários *softwares* foram utilizados. São eles: o Arduino IDE, versão 1.8.9, para a programação do Arduino; o Python, versão 3.7.3, para a programação do *script* Python; o UwAmp, versão 3.1.0, para a criação do servidor local; o HeidiSQL, versão 10.1.0, para a criação do banco de dados; e o NetBeans IDE, versão 11.0, para a programação da interface. A máquina utiliza o sistema operacional Windows 10, com 1 TB de HD, 8 GB de memória RAM, processador Intel Core i7 8ª geração e placa de vídeo NVIDIA GeForce MX150 de 2 GB.

Dessa maneira, o desenvolvimento foi dividido em cinco etapas: a definição da rede de Petri, a programação do Arduino (PNRD), a programação do *script* Python, a criação do banco de dados (MySQL) e a programação da interface (PHP e HTML). Essas etapas serão detalhadas a seguir.

3.1 Definição da Rede de Petri

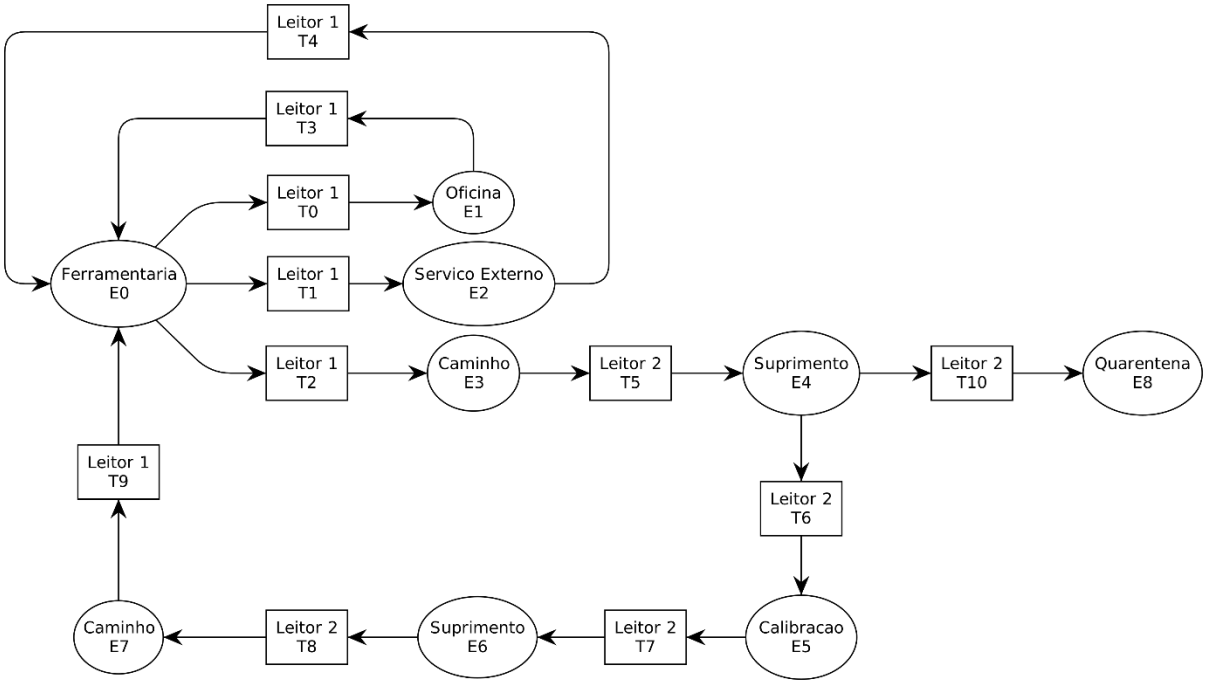
Como ponto inicial para definição da rede de Petri, o objeto a ser controlado deve ser definido. O objetivo do sistema desenvolvido é controlar a movimentação de ferramentas de uma empresa, que podem ser classificadas em três tipos: especiais, comuns e calibráveis. Porém, o foco inicial do projeto está apenas no último tipo, as ferramentas calibráveis.

A partir da definição do objeto, o seu comportamento deve ser modelado. Inicialmente, essas ferramentas se encontram em um lugar físico apropriado, a Ferramentaria. Suas possíveis movimentações podem ser divididas em três outros lugares: a Oficina, o Serviço Externo e o Suprimento. Nos dois primeiros, a ferramenta somente sai e retorna para a Ferramentaria, não passando por nenhum outro local neste percurso. Porém, ao sair para o Suprimento, a ferramenta passa por um caminho de transição, podendo então seguir por outros dois lugares, a Quarentena e a Calibração. Caso entre para a Quarentena, a ferramenta fica em um estado no

qual não pode mais ser utilizada, a não ser que uma ação externa seja executada. Já quando a ferramenta segue para Calibração, posteriormente ela retorna ao Suprimento e, finalmente, passando pelo caminho (E7), chega novamente à Ferramentaria.

Além da definição dos locais de interesse, os quais estão vinculados aos lugares da rede de Petri, as condições para que a ferramenta se movimente de um local para outro também precisam ser definidas. Essas condições, por sua vez, estão vinculadas às transições da rede de Petri, e com a adição da tecnologia RFID, elas se relacionam diretamente com os leitores. Para o estudo de caso em questão, foi proposta a utilização de dois leitores, um na entrada/saída da Ferramentaria e outro na entrada/saída do Suprimento. Dessa forma, todas as transições estão relacionadas a um leitor, de maneira que a posição das *tags* possa ser atualizada quando houver uma movimentação. Assim, a representação da rede de Petri para o estudo de caso em questão está ilustrada na Figura 14.

Figura 14 – Rede de Petri do estudo de caso em questão



Fonte: própria autora.

Nota-se na Figura 14 que, para sair da Ferramentaria, uma condição extra deve ser dada pelo usuário do sistema, que consiste na escolha do local de destino da ferramenta. Apenas após essa escolha uma das três transições (T0, T1 ou T2) é habilitada, possibilitando o Leitor 1 atualizar corretamente a posição da etiqueta relacionada à ferramenta. Para sair do Suprimento (E4), uma condição extra também é necessária. Para isso, um botão foi utilizado, de maneira que quando a etiqueta passa pelo Leitor 2 e o botão é acionado, o disparo da transição T10 é efetuado, e caso não seja acionado, dispara-se a transição T6.

Dessa forma, a rede de Petri da Ferramentaria possui nove lugares (linhas) e onze transições (colunas), e a matriz de incidência deste caso pode ser definida como:

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

Considerando que inicialmente todas as ferramentas se encontram no lugar E0 (Ferramentaria), o vetor de marcações para cada ferramenta, pode ser definido como:

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

Assim, dependendo da posição da ferramenta, certas transições serão habilitadas, e o vetor de disparos é atualizado conforme a ferramenta se movimenta pelo ambiente, ou o *token* pela rede. Vale ressaltar que cada ferramenta tem a sua própria rede de Petri, pois a matriz de incidência e o vetor de marcações serão gravados em cada etiqueta. Portanto, cada *token* opera independentemente, e a posição do conjunto só será conhecida pelo armazenamento das informações no banco de dados.

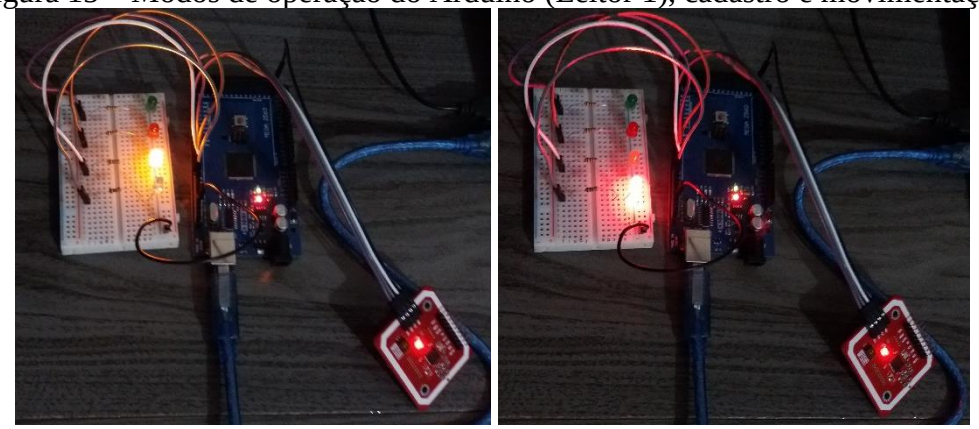
3.2 Programação do Arduino

Para a programação do Arduino, utilizou-se a biblioteca desenvolvida por Silva (2017) e seguiu-se o exemplo de PNRD clássica apresentado em seu trabalho. Toda a estrutura foi aproveitada, realizando apenas as adaptações necessárias para o caso da Ferramentaria. A rede de Petri foi configurada conforme descrito no tópico anterior e os disparos foram estruturados de forma sequencial, com as condições da posição da etiqueta no momento e do lugar escolhido pelo usuário, quando necessário. Outra modificação realizada foi a forma de leitura da etiqueta.

Na biblioteca desenvolvida por Silva (2017), a função *getTagID()* só retorna 1 *byte* da UID da *tag*, o que não é viável para o caso estudado, pois algumas etiquetas apresentam valores de identificação iguais, quando lidas dessa forma. Assim, optou-se por utilizar a função *read()* da biblioteca do leitor (Wilson, 2018), que retorna toda a UID da *tag* (4 *bytes*, no caso em questão).

Para cada *tag*, existem duas lógicas de operação a serem executadas: a gravação da rede de Petri e a execução dos disparos. Inicialmente, essas duas lógicas de operação foram tratadas separadamente. Porém, levando em consideração a execução do sistema no dia a dia da empresa, percebeu-se que novas etiquetas podem ser gravadas a qualquer momento. Portanto, não é viável haver essa troca de lógica com grande frequência, visto que o usuário do sistema poderá não compreender a lógica de programação do Arduino. Assim, os dois códigos foram unidos, e uma condição vinda da interface solicita a escolha de operação do Arduino, que poderá ser "cadastro" ou "movimentação". Cada um desses modos é representado pelo acionamento de um LED, de cor amarela para o primeiro e de cor vermelha (LED transparente) para o segundo, como apresentado na Figura 15.

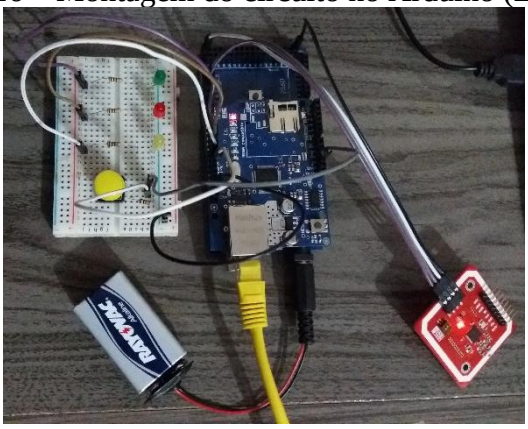
Figura 15 – Modos de operação do Arduino (Leitor 1), cadastro e movimentação



Fonte: própria autora.

Para a utilização de dois leitores em pontos distintos, dois Arduinos tornam-se necessários. Dessa maneira, o Leitor 1, relacionado às transições ligadas à Ferramentaria, executa os disparos T0, T1, T2, T3, T4 e T9, enquanto o Leitor 2, relacionado às transições ligadas ao Suprimento, executa os disparos T5, T6, T7, T8 e T10. Além dessa diferença, somente a lógica de programação do Leitor 1 possui o modo de "cadastro" da rede de Petri na etiqueta. A montagem do Leitor 2, com o *Shield Ethernet*, se dá como ilustrado na Figura 16.

Figura 16 – Montagem do circuito no Arduino (Leitor 2)



Fonte: própria autora.

Nos dois casos, o LED verde representa uma leitura/gravação realizada com sucesso e o vermelho representa um erro de leitura ou uma leitura efetuada no local inadequado (por exemplo, a *tag* se encontra no Suprimento e é percebida pelo Leitor 1). No segundo caso, o LED amarelo representa a Quarentena. Quando a leitura for efetuada e o botão for pressionado, o LED acenderá e indicará que o disparo da transição T10 foi executado. Como o Leitor 2 não possui modos de operação, basta ler a etiqueta para que o disparo seja efetuado, em função da posição atual da mesma. Já no Leitor 1, a leitura só pode ser efetuada quando o Arduino entrar em um dos modos, dado que é selecionado pelo usuário da interface.

Devido ao não estabelecimento da comunicação do Arduino com o banco de dados via *Ethernet*, os testes foram executados com apenas um leitor. Os códigos do Leitor 1 e do Leitor 2 foram unidos e toda a comunicação testada foi via Serial. Neste código as bibliotecas utilizadas foram a desenvolvida por Silva (2017) e a do leitor PN532. Vale ressaltar que neste caso, como os dados trafegam apenas via Serial, a comunicação do Arduino IDE com o sistema RFID foi definida como a Serial1, enquanto a comunicação do Arduino IDE com o *script* Python foi definida como a Serial0. A versão final deste código se encontra no Apêndice A.

Para o código do Leitor 2, além das duas bibliotecas anteriormente citadas, são necessárias também a biblioteca de conexão com o banco de dados MySQL (Bell, 2019) e a biblioteca do *Shield Ethernet*, que já vem instalada no Arduino IDE. Neste caso, apenas a Serial1 é utilizada para a comunicação com o sistema RFID, uma vez que os dados seriam enviados para o banco de dados via *Ethernet*.

3.3 Programação do *Script* Python

O *script* Python foi necessário para estabelecer a comunicação entre o Arduino IDE e o servidor *web*, e o código para esta aplicação se encontra no Apêndice B. Dessa forma, toda a comunicação envolvendo o Leitor 1 se dá via Serial. A programação em Python é responsável por três tarefas:

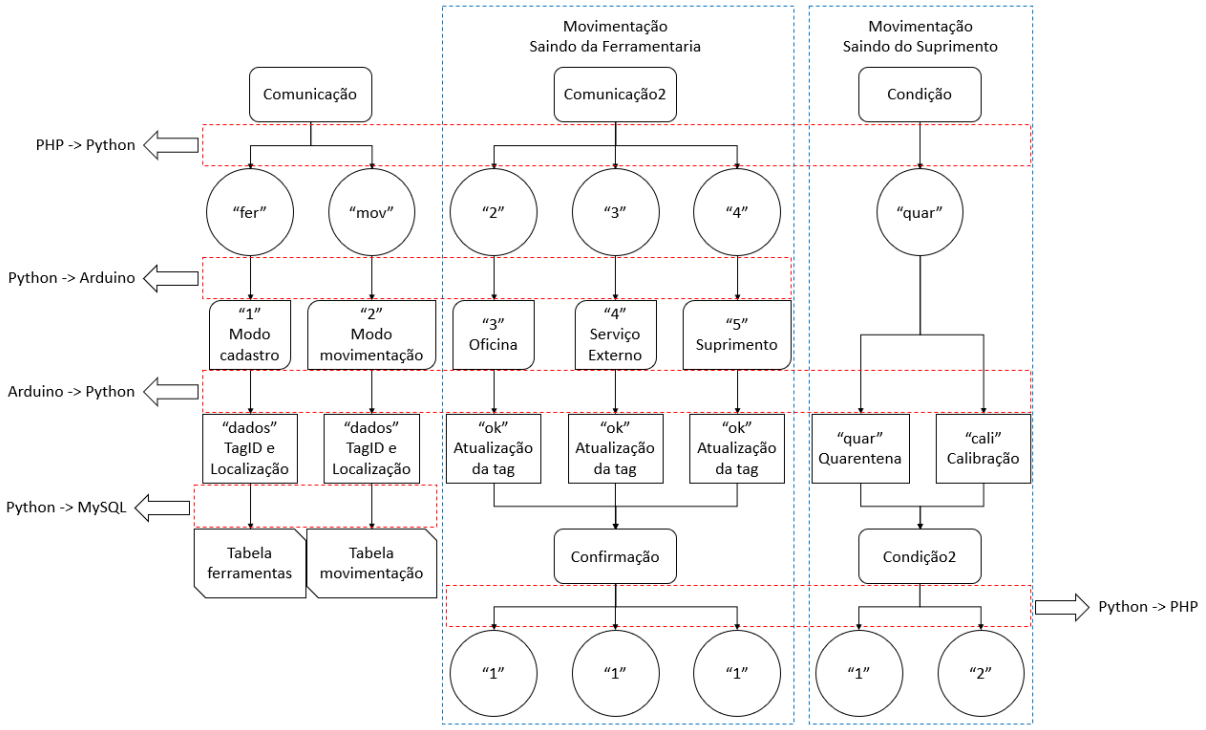
- Receber os dados de leitura das etiquetas do Arduino e transportá-los para o banco de dados MySQL;
- Ler os dados que o PHP envia para um arquivo texto e enviá-los para o Arduino, com o intuito de definir o seu modo de operação ou enviar a condição necessária para escolher qual transição disparar ao sair da Ferramentaria;
- E receber dados de confirmação de atualização da *tag* e de qual disparo foi efetuado na saída do Suprimento (T6 ou T10), e enviá-los para um arquivo texto, para o PHP atualizar diretamente as informações no banco de dados.

Dessa forma, existem cinco arquivos de texto, sendo que três deles são responsáveis por enviar informações do PHP para o *script* Python e os outros dois são responsáveis pela movimentação dos dados no sentido oposto. Os três primeiros arquivos e suas funções são: “comunicação” (envia o modo de operação), “comunicação2” (envia o local escolhido pelo usuário ao sair da Ferramentaria) e “condição” (envia uma condição para manter o *script* Python em um *loop*). Os outros dois arquivos e suas funções são: “confirmação” (envia a confirmação da atualização da etiqueta) e “condição2” (envia o local escolhido pelo usuário ao sair do Suprimento).

A operação do *script* está representada na Figura 17. Os quadros vermelhos englobam as transferências de dados que envolvem o *script* Python, podendo este receber ou enviar informações para o Arduino, PHP ou banco de dados. Os quadros azuis separam os casos de movimentações saindo da Ferramentaria e saindo do Suprimento. As figuras geométricas são:

- Retângulos com cantos arredondados: arquivos de texto;
- Círculos: conteúdo dos arquivos de texto;
- Retângulos com dois cantos arredondados: informações enviadas para o Arduino;
- Retângulos: informações enviadas pelo Arduino;
- Retângulos com dois cantos chanfrados: tabelas do banco de dados.

Figura 17 – Operação do *script* Python



Fonte: própria autora.

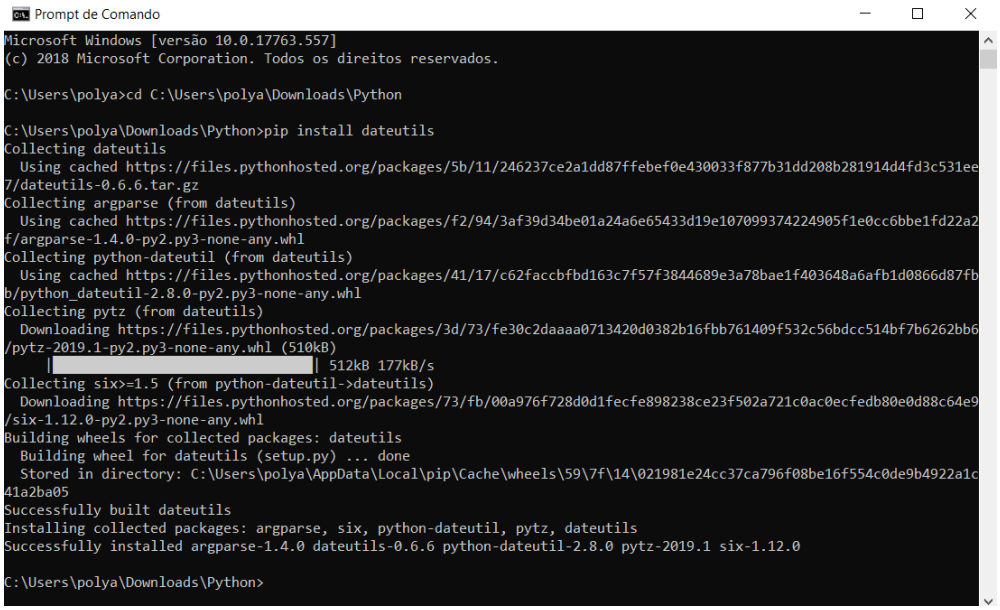
A operação ocorre da seguinte forma: primeiramente espera-se algum dado ser escrito no arquivo texto "comunicação" que o PHP gera. Ao receber esse dado, o *script* Python o envia pela porta Serial para o Arduino, que entra em algum modo de operação e espera a leitura de uma *tag*. Quando isso ocorre, o Arduino envia os dados de ID e de localização da etiqueta para o *script* Python, o qual insere esses dados no banco de dados MySQL. Após isso, o *script* Python abre novamente o arquivo texto "comunicação" e exclui os dados nele contidos, evitando que uma nova leitura sem intenção seja cadastrada no banco de dados.

Caso seja o cadastro de movimentação para quando a ferramenta sai da Ferramentaria, onde é necessária uma escolha do usuário, a operação é um pouco diferente. A parte inicial é idêntica, porém, quando a *tag* é lida, o Arduino entra em modo de espera até que a informação do local de destino da ferramenta seja recebida. Essa informação é enviada pelo PHP para o arquivo texto "comunicação2", em um momento posterior. Da mesma maneira, ao receber esse dado, o *script* Python o envia para o Arduino, que só então é capaz de efetuar um disparo na rede. Ao atualizar a *tag* corretamente, o Arduino envia essa confirmação para o *script* Python, que a envia para o PHP no arquivo texto "confirmação", realizando o cadastro da movimentação. Após isso, o *script* Python novamente abre os arquivos e deleta as informações neles presentes.

Para o caso em que a ferramenta sai do Suprimento e pode seguir por dois caminhos, Quarentena e Calibração, após ler a etiqueta, o PHP envia no arquivo texto “condição” um dado para gerar um *loop* no *script* Python, que aguarda um dado vindo do Arduino sobre qual transição foi disparada. Ao receber esse dado, o script Python escreve no arquivo texto “condição2”, que é lido pelo PHP para cadastrar a informação correta no banco de dados. Após isso, o *script* Python novamente abre os arquivos e deleta as informações neles contidas. Vale ressaltar que essas exclusões de dados também são efetuadas pelo PHP quando a interface retorna ao menu de movimentações, para garantir que nenhum cadastro indesejado seja efetuado.

Para a criação deste código, três bibliotecas do Python foram necessárias: *time*, *serial* e *mysql.connector*. A primeira é uma biblioteca padrão de tempo, que já vem instalada com o programa. As outras duas foram instaladas com o auxílio do *pip*, um sistema de gerenciamento de pacotes usado para instalar e gerenciar pacotes de *software* escritos na linguagem de programação Python. Para isso, utiliza-se o comando “*pip install*” no *prompt* de comando do Windows, com o diretório da pasta onde está instalado o Python, que a biblioteca desejada é instalada. A Figura 18 apresenta a instalação da biblioteca *dateutils*, que contém extensões da biblioteca básica de data e tempo.

Figura 18 – Instalação da biblioteca *dateutils* pelo comando “*pip install*”



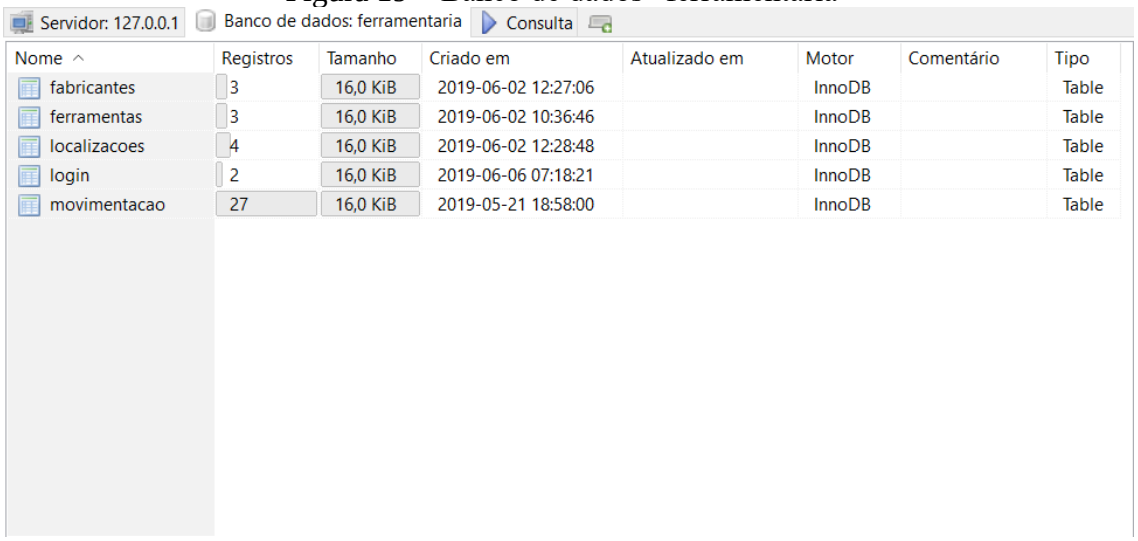
Fonte: própria autora.

3.4 Criação do banco de dados

O banco de dados é um conjunto de registros sobre pessoas, lugares ou objetos. Essas informações podem ser divididas em tabelas, e cada tabela possui suas colunas, sendo cada coluna um tipo de dado. Esses dados podem ser de tipos diferentes, como um texto, um número inteiro ou um número decimal.

Para o estudo de caso em questão, criou-se o banco de dados “ferramentaria”, com o auxílio do programa HeidiSQL. Dentro desse banco de dados, cinco tabelas foram criadas, sendo elas: fabricantes, localizações, *login*, ferramentas e movimentação. A Figura 19 apresenta o banco de dados criado.

Figura 19 – Banco de dados “ferramentaria”



Nome	Registros	Tamanho	Criado em	Atualizado em	Motor	Comentário	Tipo
fabricantes	3	16,0 KiB	2019-06-02 12:27:06		InnoDB		Table
ferramentas	3	16,0 KiB	2019-06-02 10:36:46		InnoDB		Table
localizacoes	4	16,0 KiB	2019-06-02 12:28:48		InnoDB		Table
login	2	16,0 KiB	2019-06-06 07:18:21		InnoDB		Table
movimentacao	27	16,0 KiB	2019-05-21 18:58:00		InnoDB		Table

Fonte: própria autora.

Em todas as tabelas é necessário um código definido como chave primária e auto incremental, para que haja um controle das linhas em que os dados são inseridos, o que serve de base para a programação do PHP. Portanto, a primeira coluna de todas as tabelas é o código incremental.

As tabelas de fabricantes e localizações são mais simples, e possuem apenas duas colunas cada, sendo a segunda coluna, fabricante e localização, respectivamente. Nessas tabelas ficarão cadastrados os fabricantes e as localizações para que o usuário insira os dados sempre no mesmo padrão, evitando erros. A tabela de *login* possui quatro colunas, sendo a segunda, nome, a terceira, usuário e a quarta, senha. Nessa tabela serão registrados os *logins* de cada um dos usuários do sistema, assim como suas senhas. Nenhuma proteção para as senhas foi utilizada, pois o sistema ainda se encontra em fase de desenvolvimento. Porém, a segurança dos dados do sistema é sempre prioridade e isso deve ser implantando futuramente.

As tabelas de ferramentas e de movimentação são as mais importantes do banco de dados. Na primeira, serão cadastrados todos os dados das ferramentas, como nome, modelo, fabricante, número de série, número de patrimônio, faixa de indicação, data de calibração, localização, tipo, foto e a ID da *tag* que será fixada na ferramenta. Essa tabela servirá para controlar a quantidade de ferramentas que a empresa possui. Já na tabela de movimentação, serão cadastrados o código da ferramenta (para adquirir na tabela de ferramentas o modelo e o fabricante da *tag* lida), a ID da *tag*, a localização, o destino, o nome do mecânico, a ordem de serviço, a data e a hora da movimentação. Essa tabela tem a função de controlar toda a movimentação das ferramentas com o auxílio da tecnologia RFID, que é o objetivo principal deste trabalho.

3.5 Programação da interface

A programação da interface foi feita em linguagem PHP e HTML, utilizando o software NetBeans IDE. Para o desenvolvimento de elementos de interface foi utilizado o *bootstrap* (Bootstrap, 2019), um *framework web* com código-fonte aberto para sites e aplicações *web* usando HTML. E, para o funcionamento do *bootstrap*, o *jquery* (jQuery, 2019), uma biblioteca de funções JavaScript que interage com o HTML, é necessário. Outro recurso utilizado foi o *bootbox* (Bootbox, 2019), uma biblioteca JavaScript que simplifica a criação de caixas de diálogos usando o *bootstrap*.

Dois exemplos de estilo de páginas do *bootstrap* foram utilizados, o *dashboard* (Bootstrap, 2019a) e o *sign in* (Bootstrap, 2019b). Em ambos os casos, o código-fonte dos exemplos foi copiado e as devidas alterações foram executadas.

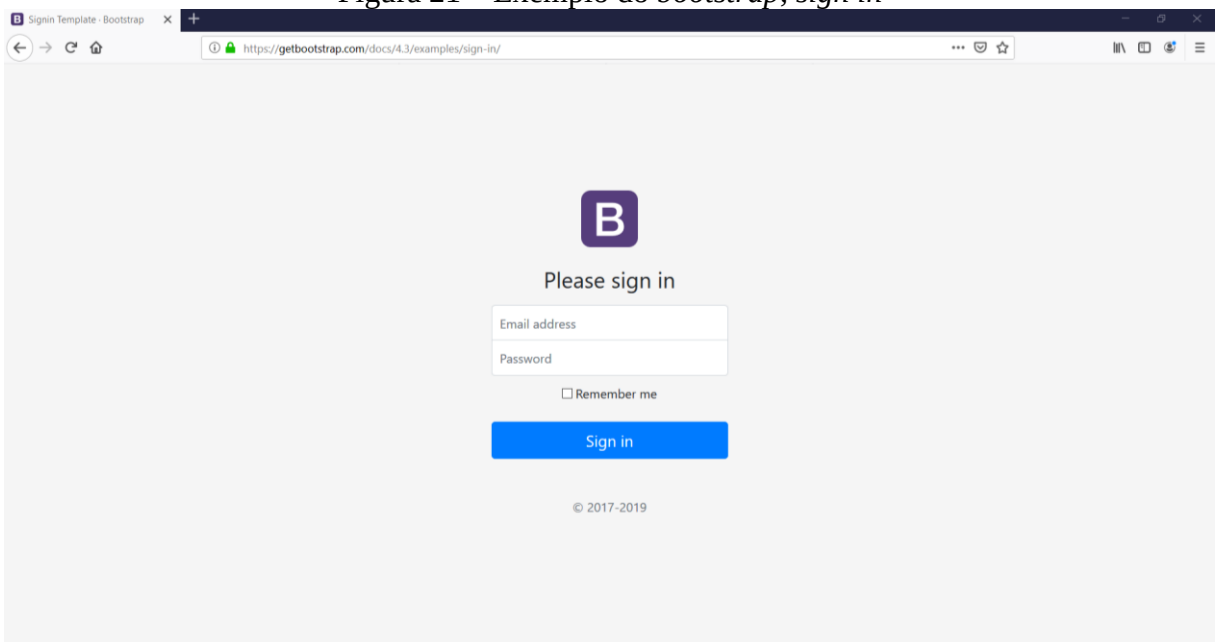
O *dashboard* foi utilizado como a base da interface, e está ilustrado na Figura 20. A logo da empresa foi inserida no canto superior esquerdo, a barra de procura foi substituída por um título, o menu na esquerda foi alterado conforme necessário e na tela foi utilizado apenas a tabela, sem o gráfico. O texto foi escrito em português e alguns fatores da aplicação foram adicionados, como por exemplo, filtros de dados. O resultado da interface será apresentado na próxima seção.

Figura 20 – Exemplo do *bootstrap*, *dashboard*

Fonte: Bootstrap, 2019a.

O *sign in* foi utilizado como tela de *login* do sistema desenvolvido, e está apresentado na Figura 21. As alterações nessa tela foram somente a logo da empresa e a exclusão da caixa de “Remember me” e da data abaixo do botão. O texto também foi escrito em português e um botão para visualização da senha foi adicionado. Essa tela pode ser vista na seção de resultados.

Figura 21 – Exemplo do *bootstrap*, *sign in*



Fonte: Bootstrap, 2019b.

Dessa forma, o HTML, juntamente com o CSS (*Cascading Style Sheets*, ou Folha em Estilo Cascata) são os responsáveis pela estilização da página. A função do PHP na

programação da interface é fazer toda a ligação com o banco de dados, através de funções mysql disponíveis na linguagem. É dessa forma que as tabelas da interface são preenchidas com os dados do sistema RFID, uma vez que estes são gravados no banco de dados através do *script* Pyhton ou da conexão *Ethernet*. O PHP também executa validações de *login* e é o responsável por gerar o arquivo texto que envia as informações sobre a escolha do usuário para o *script* Python, dado que é posteriormente enviado ao Arduino. Os arquivos desenvolvidos estão disponíveis no GitHub (Alves, 2019).

4 RESULTADOS

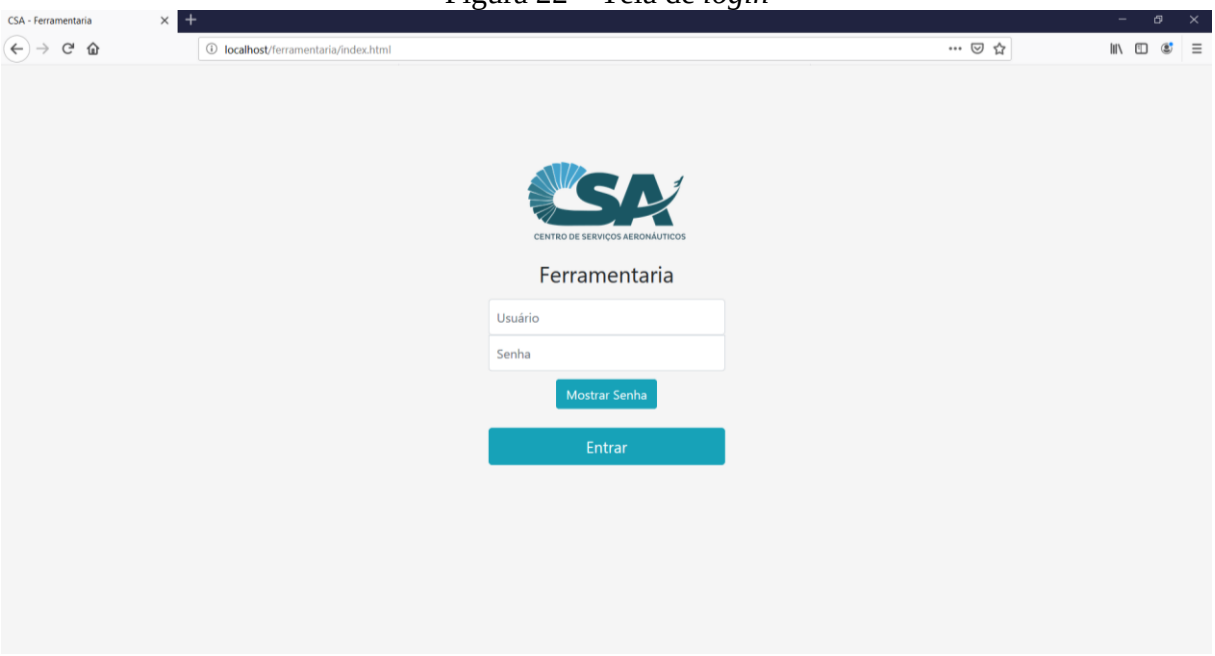
A intenção inicial do projeto era utilizar dois leitores, um na Ferramentaria e outro no Suprimento. Porém, devido ao pouco conhecimento em banco de dados, as configurações de permissões de acesso não foram alteradas, não permitindo ao Arduino enviar informações para o banco de dados. Dessa forma, apenas o Leitor 1 foi utilizado para testes, contento toda a programação da rede. A adição do Leitor 2 no projeto ficou definida como trabalho futuro.

Os resultados deste trabalho estão voltados principalmente para a interface, e como ela se comunica com o sistema RFID. Duas tabelas do banco de dados são responsáveis por armazenar as informações das etiquetas, e cada uma terá o seu menu na interface, porém, a API também possui outras aplicações, como o cadastro de usuários, de fabricantes, de localizações e a possibilidade de gerar relatórios. Essas aplicações serão detalhadas a seguir.

4.1 Tela de *login*

A tela de *login* foi baseada no exemplo *sign in* do *bootstrap*. Essa tela foi adicionada para existir um controle sobre quem pode acessar o sistema e com quais permissões, evitando erros de utilização da interface por usuários não treinados. A Figura 22 ilustra o resultado dessa tela inicial.

Figura 22 – Tela de *login*



Fonte: própria autora.

4.2 Menu de ferramentas

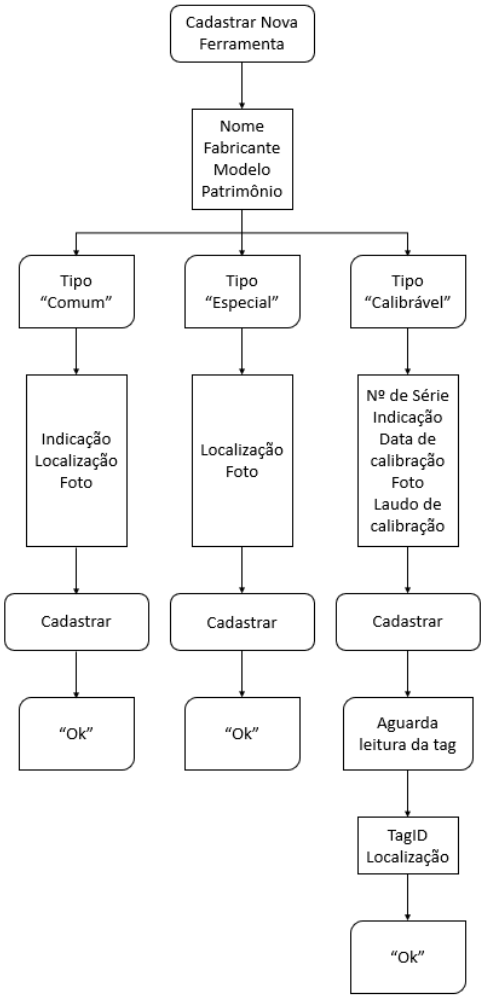
As informações contidas na tabela de ferramentas do banco de dados estarão presentes no menu de ferramentas. As colunas de TagID e de Localização são dados preenchidos pelo sistema RFID, enquanto nas demais os dados são preenchidos pelo usuário do sistema, no ato de cadastro de uma nova ferramenta. Os ícones da câmera e da lista representam a foto e o laudo de calibração, respectivamente. Cada linha possui as funções de alterar (ícone do lápis) e excluir (ícone do lixo), caso seja necessário corrigir alguma informação ou excluir alguma ferramenta que foi cadastrada de maneira errada. Esse menu possui também um filtro, para localizar uma ferramenta de forma mais fácil, através do nome, do modelo, do tipo ou da localização. A Figura 23 apresenta o menu de ferramentas.

Figura 23 – Menu de ferramentas

Fonte: própria autora.

A sequência de operação para o cadastro de uma nova ferramenta está representada na Figura 24. Ao clicar no *link* “Cadastrar Nova Ferramenta”, o sistema direciona para outra página, onde as informações devem ser preenchidas pelo usuário. Alguns campos, como o nome, são obrigatórios, e caso o usuário tente cadastrar uma nova ferramenta sem preenchê-los, um aviso do tipo “O campo NOME não pode ficar em branco!” será gerado. Ao escolher o tipo de ferramenta (comum, especial ou calibrável), alguns campos extras aparecerão para serem preenchidos.

Figura 24 – Sequência de operação para o cadastro de uma nova ferramenta



Fonte: própria autora.

O cadastro de ferramentas dos tipos comum e especial é simples e não envolve o sistema RFID. Quando as informações necessárias são preenchidas e o botão "Cadastrar" é pressionado, o cadastro é realizado com sucesso.

Para o principal caso estudado, as ferramentas calibráveis, os campos a serem preenchidos são: nome, fabricante, modelo, número de patrimônio, número de série, faixa de indicação, data de calibração, foto e laudo de calibração. Esses campos estão apresentados na Figura 25.

Para maior segurança, são geradas cópias da foto e do laudo de calibração da ferramenta, para que fiquem armazenadas em uma pasta do sistema. Essas cópias são renomeadas automaticamente em função do número de patrimônio, que é algo único e presente em todas as ferramentas.

Figura 25 – Cadastro de uma nova ferramenta calibrável

Fonte: própria autora.

Após preencher as informações necessárias e ao clicar no botão “Cadastrar”, o sistema direciona para uma página que fica aguardando a leitura da *tag*. É nesse momento que o PHP gera o arquivo texto “comunicação”, enviando a informação do modo de cadastro de ferramentas para o *script* Python, e, consequentemente, para o Arduino. Assim, ao ler a etiqueta, a informação chega no banco de dados e o PHP confirma o recebimento, finalizando o cadastro da nova ferramenta. Essa tela de espera está ilustrada na Figura 26.

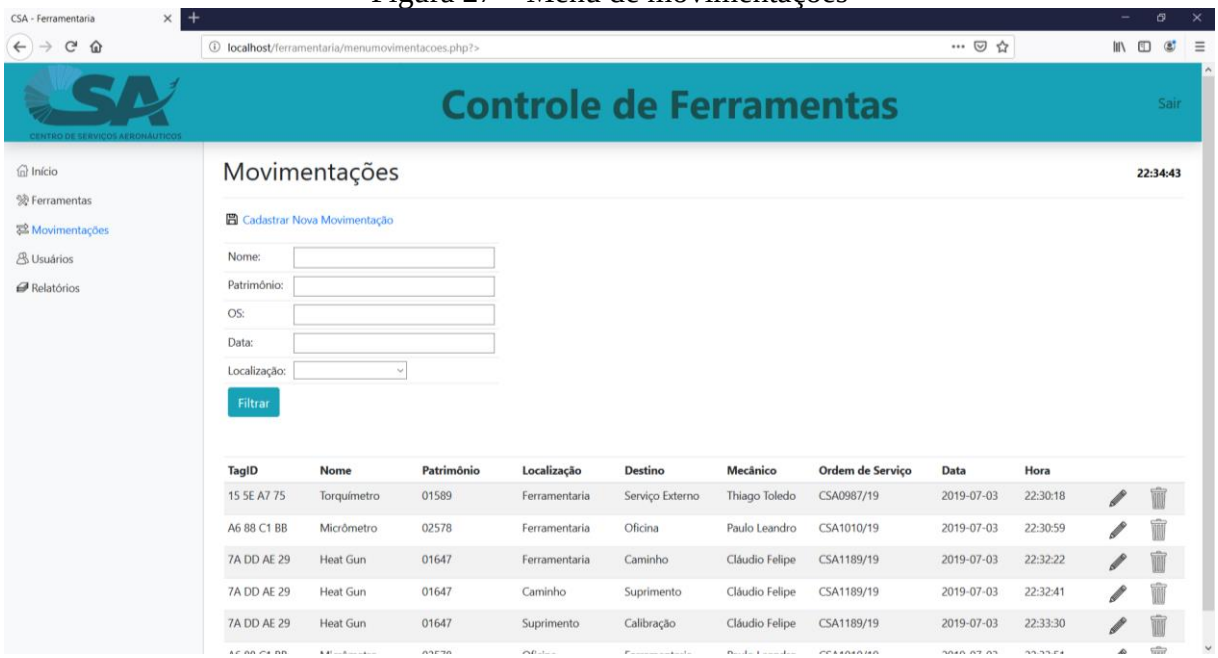
Figura 26 – Tela de espera da leitura da etiqueta

Fonte: própria autora.

4.3 Menu de movimentações

Pode-se dizer que o menu de movimentações é a parte mais importante do sistema desenvolvido, pois é o que mais interage com o sistema RFID. Essa tela é bem parecida com o menu de ferramentas, porém, apresenta as informações da tabela de movimentação do banco de dados. A primeira e a quarta colunas são preenchidas com as informações vindas do sistema RFID, enquanto o nome e o número de patrimônio são adquiridos da tabela de ferramentas mostrada anteriormente a partir da ID da etiqueta lida no momento. O destino da ferramenta, a ordem de serviço associada e o nome do mecânico responsável são dados preenchidos pelo ferramenteiro. Por fim, a data e a hora são automaticamente preenchidas pelo PHP quando o cadastro for efetuado. A Figura 27 apresenta esse menu de movimentações.

Figura 27 – Menu de movimentações



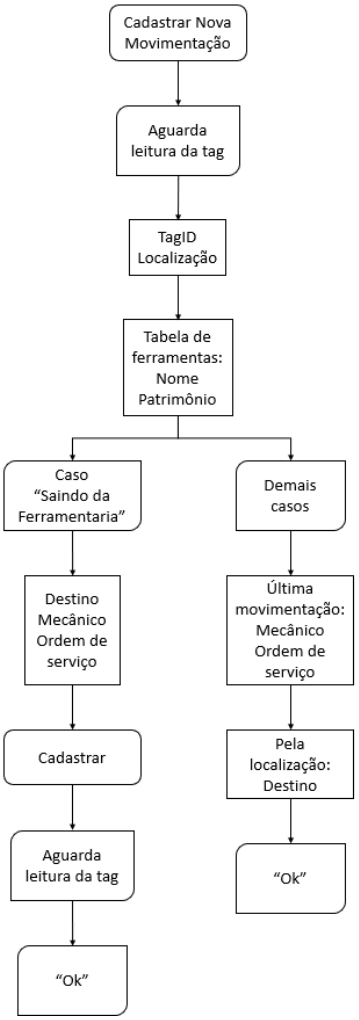
Fonte: própria autora.

Da mesma forma que o menu de ferramentas, o menu de movimentações também possui os recursos de alteração e exclusão das informações da tabela. O filtro nesse caso pode ser em função do nome ou do número de patrimônio da ferramenta, da ordem de serviço, da data ou da localização.

O cadastro de uma nova movimentação funciona de uma maneira um pouco diferente do cadastro de ferramenta. A Figura 28 ilustra a sequência de operação para todos os casos de cadastro de uma nova movimentação. Ao clicar no *link*, o sistema direciona primeiramente para a leitura da *tag*, tela semelhante ao cadastro de ferramentas, e, nesse momento, o PHP escreve no arquivo texto “comunicação” a informação do modo de operação, possibilitando a leitura da

etiqueta. Após a leitura, caso a ferramenta esteja na Ferramentaria, o sistema encaminha para uma interface onde os dados da movimentação devem ser preenchidos, sendo que a TagID e a Localização já estarão preenchidas pelos dados lidos e o Nome e o Patrimônio serão adquiridos da tabela de ferramentas.

Figura 28 – Sequência de operação para o cadastro de uma nova movimentação



Fonte: própria autora.

Ao preencher as demais informações e clicar no botão "Cadastrar", o PHP cria o arquivo texto "comunicação2", enviando para o *script* Python o destino da ferramenta, e ao mesmo tempo direciona para a tela de aguardo da leitura da *tag*. Dessa forma, ao ler a etiqueta, o Arduino é capaz de fazer o disparo correto na rede de Petri, atualizando a informação de localização da *tag*. Nesse momento, o Arduino envia uma confirmação de atualização para o script Python, que escreve no arquivo texto "confirmação" para que o PHP leia e confirme o cadastro da movimentação. Essa tela de cadastro de movimentações está apresentada na Figura 29.

Figura 29 – Cadastro de uma nova movimentação

Fonte: própria autora.

As demais movimentações, diferentemente do caso saindo da Ferramentaria, são automáticas. Ao clicar no *link* “Cadastrar Nova Movimentação”, o sistema também direciona para a tela de aguardo de leitura da *tag*. Porém, quando uma etiqueta é lida, em função da sua localização, o próximo disparo já é automaticamente efetuado e o PHP confirma a movimentação, buscando os demais dados do último cadastro daquela *tag*. Até mesmo a movimentação da saída do Suprimento é automática, pois a condição nesse caso é física (botão), não necessitando de uma escolha pela interface.

4.4 Menu de relatórios

O menu de relatórios é responsável por gerar documentos sobre as movimentações de ferramentas, e está apresentado na Figura 30. Esse tipo de documento é importante para essa aplicação de rastreabilidade de ativos, pois diversas informações podem ser adquiridas, como por exemplo as movimentações de um dia, as movimentações de certa ferramenta ou as ferramentas utilizadas dentro de uma ordem de serviço. Por se tratar de itens calibráveis, é importante confirmar que as ferramentas estavam em condições aceitáveis nos serviços executados.

Figura 30 – Menu de relatórios

Fonte: própria autora.

As movimentações podem ser escolhidas por ferramenta (número de patrimônio) ou por ordem de serviço. Ao clicar no botão “Gerar PDF”, uma nova aba é aberta, contendo um arquivo PDF com as movimentações da ferramenta escolhida ou as ferramentas utilizadas na ordem de serviço em questão. O relatório das ferramentas movimentadas na ordem de serviço CSA1010/19 está ilustrado na Figura 31. Dessa forma, é possível imprimir ou salvar o relatório, podendo ser utilizado como documento na ordem de serviço.

Figura 31 – Exemplo de relatório de movimentações por ordem de serviço

Fonte: própria autora.

4.5 Outras funções – usuários, fabricantes e localizações

Além dos dois menus mais importantes, o sistema desenvolvido possui outras funções. Existe um menu de usuários, no qual é possível visualizar todos os usuários do sistema, fazer alterações e exclusões dos dados, e cadastrar novos usuários. Porém, apenas o Administrador terá a permissão para executar essas atividades. A Figura 32 ilustra esse menu de usuários.

Figura 32 – Menu de usuários

Fonte: própria autora.

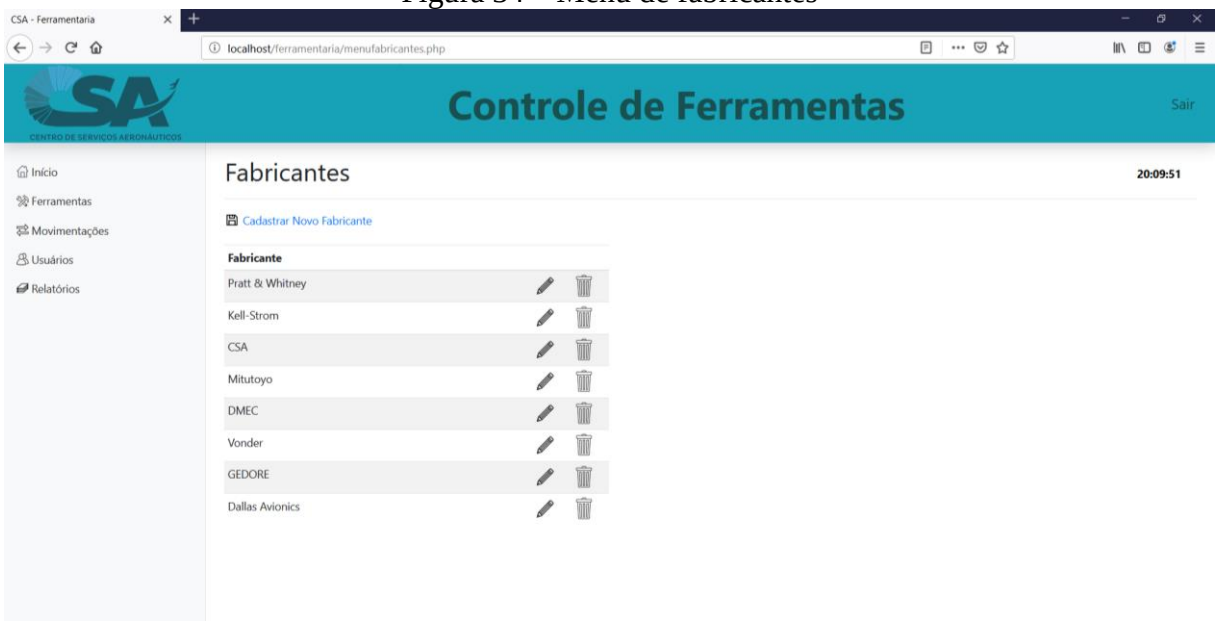
O cadastro de um novo usuário é bem mais simples e não tem relação com o sistema RFID, envolvendo apenas o PHP e o banco de dados. Basta preencher os campos de Nome, Usuário e Senha, e ao clicar no botão “Cadastrar”, as informações são inseridas no banco de dados. No campo Senha a informação é protegida, porém existe um botão que a torna visível, assim como na tela de *login*. A Figura 33 apresenta essa tela de cadastro de usuários.

Figura 33 – Cadastro de um novo usuário

Fonte: própria autora.

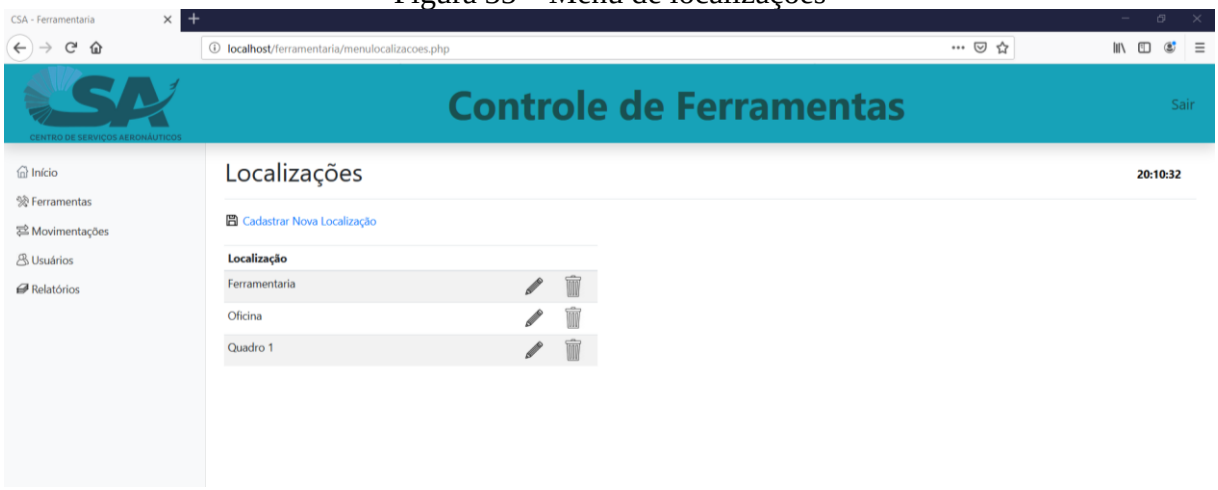
A API possui também mais dois menus, que não se encontram na lista de menus à esquerda da interface. Estes estão relacionados às outras duas tabelas anteriormente citadas do banco de dados, a de fabricantes e a de localizações, e podem ser acessadas pelo cadastro de ferramentas. Quando, ao cadastrar uma nova ferramenta, o fabricante desta ou a sua localização não existir no sistema, basta clicar no ícone do disquete ao lado do campo para que seja direcionado a esses menus. As Figuras 34 e 35 apresentam o menu de fabricantes e o de localizações, respectivamente.

Figura 34 – Menu de fabricantes



Fonte: própria autora.

Figura 35 – Menu de localizações

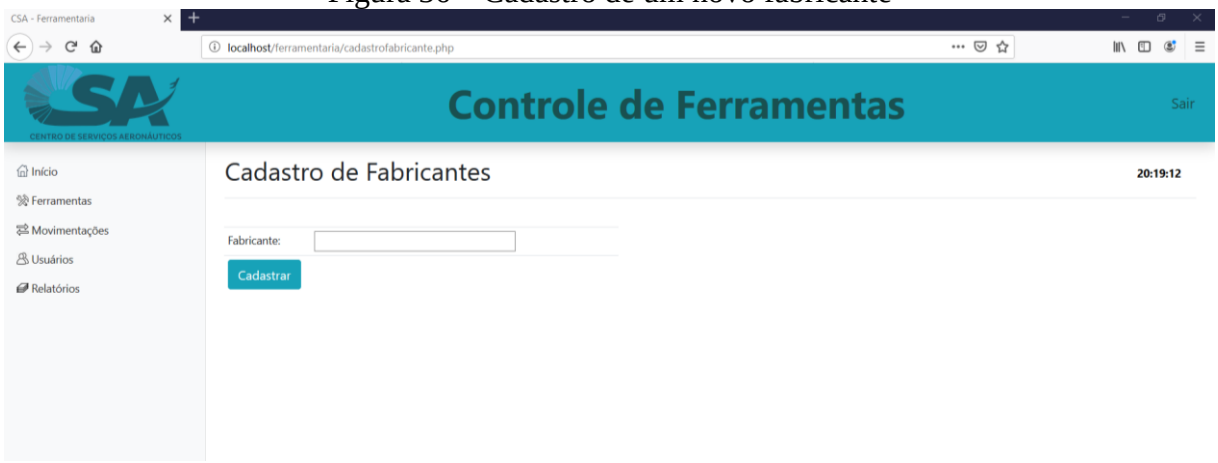


Fonte: própria autora.

Nota-se que as funções de alteração e exclusão dos dados da tabela também estão presentes nesses menus. E, da mesma maneira que no cadastro de usuários, os cadastros de

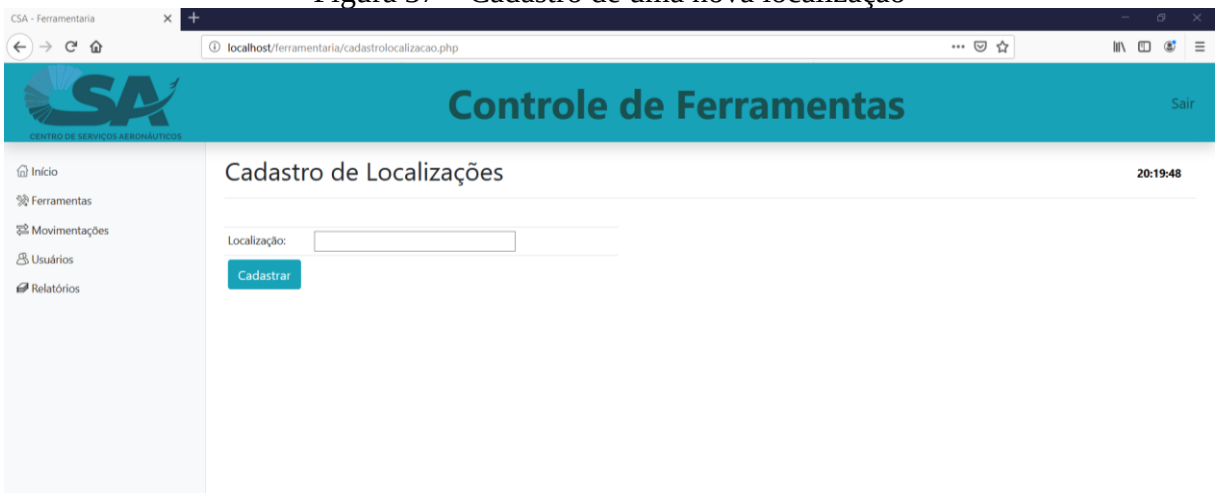
fabricantes e localizações envolvem apenas o PHP e o banco de dados. Porém, nesses casos, apenas uma informação é necessária. Essas telas estão ilustradas nas Figuras 36 e 37.

Figura 36 – Cadastro de um novo fabricante



Fonte: própria autora.

Figura 37 – Cadastro de uma nova localização



Fonte: própria autora.

5 CONCLUSÕES

Este trabalho envolveu um tema muito discutido no setor industrial atualmente, o conceito de Indústria 4.0. Noções sobre o assunto são abordadas em disciplinas de manufatura e automação do curso de Engenharia Mecânica. O trabalho desenvolvido também contou com diversos assuntos não abordados durante o curso de graduação, como a tecnologia RFID, as redes de Petri, a PNRD e, principalmente, as linguagens de programação que foram utilizadas. Estas últimas demandaram razoável esforço para o desenvolvimento do projeto, pois quando os exemplos utilizados não retornavam os resultados esperados, muito tempo era necessário para encontrar a solução.

O trabalho apresentou o desenvolvimento de uma interface de controle de ferramentas, através da modelagem do comportamento do objeto de estudo em uma rede de Petri, da identificação das ferramentas com as *tags* RFID, da aplicação da tecnologia PNRD embarcada em Arduino, do armazenamento de informações em um banco de dados, da integração desses elementos e da execução dos testes para validação do protótipo.

Primeiramente, modelou-se o comportamento das ferramentas calibráveis, identificando os possíveis lugares por onde esse tipo de objeto caminha. Essa etapa foi executada facilmente, sem muitos problemas com relação à falta de conhecimento sobre o assunto. Poucas pesquisas sobre as redes de Petri foram suficientes para aprender seus conceitos básicos, sendo possível elaborar uma rede simples e aplicar sua definição matemática, obtendo a matriz de incidência e os vetores de marcações e de disparos corretamente.

A identificação dos objetos com o sistema RFID foi um ponto não executado totalmente na prática. O princípio de funcionamento da tecnologia foi estudado, porém, os dispositivos utilizados foram escolhidos com base no trabalho já desenvolvido por Silva (2017), e não com base na aplicação. Dessa forma, testes em ambiente real devem ser executados posteriormente, com as etiquetas fixadas nas ferramentas, sendo *tags* do tipo chaveiro ou adesivas. Isso é necessário para verificar se há possíveis erros de leitura e gravação devido ao tipo de material e meio inserido.

A aplicação da tecnologia PNRD embarcada em Arduino, através da biblioteca desenvolvida por Silva (2017), também foi uma etapa executada sem muitos problemas, uma vez que os exemplos disponibilizados pelo autor da biblioteca são claros e diretos. A linguagem de programação do Arduino é simples, facilitando o desenvolvimento das condições para cada disparo na rede de Petri. A maior dificuldade encontrada nessa etapa foi em relação à comunicação do Arduino com os outros elementos computacionais (*script* Python, banco de

dados e PHP), uma vez que foi necessário aprender a comunicar com o *script* Python e criar *loops* para que as leituras e gravações nas etiquetas ocorressem com sucesso. Alguns problemas de leitura das etiquetas também foram percebidos. Para impedir que isso atrapalhasse o funcionamento do sistema, alguns *prints* de erros foram comentados na biblioteca do leitor utilizado. Dessa forma, caso houvesse algum erro na leitura de uma etiqueta, apenas o LED vermelho acenderia, e o *loop* criado faria com que a leitura tentasse ser executada novamente, não enviando nenhum valor indesejado para o *script* Python, pois isso parava a sua execução.

A etapa de criação do banco de dados MySQL também foi executada facilmente. Apenas alguns tutoriais foram necessários para aprender a criar um banco de dados e suas tabelas, além das colunas que cada tabela possui. Para enviar os dados do sistema RFID para o banco de dados criado, a biblioteca disponível para o Python foi utilizada sem problemas, seguindo os exemplos encontrados na internet.

A etapa que apresentou maiores dificuldades para o desenvolvimento do trabalho foi a criação da interface, uma vez que esse tipo de programação não é abordado durante o curso de Engenharia Mecânica e a autora recorreu à exemplos disponíveis na internet e à ajuda de pessoas com conhecimento na área. Como a parte de estilização da interface foi baseada nos exemplos do *bootstrap*, sua utilização foi mais simples. Porém, algumas formas de definição de elementos no CSS não foram completamente entendidas. Ainda assim, a parte que mais apresentou dificuldades foi a programação do PHP. As funções utilizadas para a conexão com o banco de dados, apesar de serem simples para as pessoas da área, são confusas e exigem um pouco mais de estudo na linguagem, pelo ponto de vista da autora.

Apesar das dificuldades encontradas, as funcionalidades do sistema foram integradas e a comunicação entre os elementos utilizados foi estabelecida com sucesso. Os testes executados com o protótipo retornaram bons resultados, validando a proposta. O resultado do trabalho foi, então, satisfatório, uma vez que os objetivos específicos foram alcançados, atingindo o objetivo geral, que é o desenvolvimento de uma interface para controle de ferramentas.

Porém, algumas implementações ainda necessitam ser efetuadas, tanto na parte da tecnologia RFID, quanto na interface. Devido à baixa qualidade do *hardware* utilizado, o sistema ainda é muito passível de erro, principalmente nas leituras das etiquetas, e caso os usuários não sigam os procedimentos adequados, é fácil haver um desencontro de informações (dados da *tag* e dados cadastrados na interface).

Muitos micro e pequenos empreendedores acreditam que investir em novas tecnologias é inviável devido aos custos inerentes, porém, como visto neste trabalho, um sistema simples de controle de ativos pode ser implementado com baixo custo. Para o protótipo desenvolvido,

algo em torno de R$ 400,00 (excluindo-se o computador) seria suficiente para implementá-lo, visto que os gastos são apenas com componentes eletrônicos, uma vez que os *softwares* utilizados são todos livres. Mas, caso o sistema necessite de maior robustez e ser menos suscetível a erros, é evidente que maiores custos de *hardware* estarão envolvidos.

A intenção do projeto é deixar o sistema cada vez mais automatizado. Para isso, duas sugestões de implementações futuras são: a adição de mais leitores, para que mais caminhos possam ser controlados e para que a atualização das etiquetas aconteça com maior frequência; e a adoção de cartões RFID como crachás, possibilitando controlar o acesso à ferramentaria e, assim, não necessitando de um ferramenteiro para atualizar o sistema, pois a leitura do crachá já entraria no banco de dados com o nome do mecânico.

A maior contribuição deste trabalho está voltada para a aplicação da tecnologia PNRD, através da biblioteca desenvolvida por Silva (2017). Porém, o sistema desenvolvido integra outros conceitos, contribuindo também com o desenvolvimento de uma interface em um servidor *web* local, aliado a um banco de dados. Dessa forma, este projeto serve de base para outros trabalhos na mesma linha de aplicação, e parte dos códigos desenvolvidos pode ser utilizada. Outra sugestão de trabalho futuro seria o desenvolvimento de uma interface como aplicativo para celular, facilitando o monitoramento e o controle do sistema de lugares remotos.

REFERÊNCIAS BIBLIOGRÁFICAS

ABDI. **Indústria 4.0.** Disponível em: <http://www.industria40.gov.br/>. Acesso em 11/06/2019.

ACURA. **Leitor RFID.** Disponível em: <http://www.acura.com.br/pt/produtos/leitor>. Acesso em 02/07/2019.

ALVES, P. **Códigos desenvolvidos na criação de um software de controle de inventários, utilizando um sistema RFID.** Disponível em: <https://github.com/PollyanaAlves/ferramentaria>. Criado em 03/07/2019.

ARDUINO. **What is Arduino?** Disponível em: <https://www.arduino.cc/en/Guide/Introduction>. Acesso em 28/05/2019.

BELL, C. **Database connector library for using MySQL with your Arduino projects.** Disponível em: <https://github.com/ChuckBell/MySQL_Connector_Arduino>. Acesso em 25/06/2019.

BOOTBOX. **Getting Started.** Disponível em: <http://bootboxjs.com/getting-started.html>. Acesso em 06/05/2019.

BOOTSTRAP. **Download.** Disponível em: <https://getbootstrap.com.br/docs/4.1/getting-started/download/>. Acesso em 06/05/2019.

BOOTSTRAP. **Dashboard.** Disponível em: <https://getbootstrap.com/docs/4.3/examples/dashboard/>. Acesso em 15/06/2019a.

BOOTSTRAP. **Sign in.** Disponível em: <https://getbootstrap.com/docs/4.3/examples/sign-in/>. Acesso em 15/06/2019b.

FONSECA, J.P.S. **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.** Tese de Doutorado – Universidade Federal de Uberlândia, Minas Gerais, Brazil, 2018.

FONSECA, J.P.S., TAVARES, J.J.P.Z.S. **Petri Net with RFID Distributed Database for aous Search and Rescue in Trails and Crossings.** Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE'17), p. 229-230, 2017.

FRANCES, C.R.L. **Introdução às Redes de Petri.** Laboratório de Computação Aplicada – Universidade Federal do Pará, Brazil, 2003.

JENSEN, K., KRISTENSEN, L.M. **Coloured Petri Nets: Modelling and Validation of Concurrent Systems.** Berlin, Heidelberg: Springer-Verlag, 2009. 374 p.

JQUERY. **Downloading jQuery.** Disponível em: <https://jquery.com/download/>. Acesso em 06/05/2019.

LAGE, B. **Aprendendo a programar em Python – Introdução.** Disponível em: <https://www.devmedia.com.br/aprendendo-a-programar-em-python-introducao/17093>. Acesso em 11/06/2019.

MURATA, T. **Petri Nets: Properties, analysis and applications.** Proceedings of the IEEE. New York, v. 77, n. 4, p. 541-580. abr. 1989. Disponível em: <http://ieeexplore.ieee.org/document/24143/>. Acesso em 28/05/2019.

NETO, O.T., JUNIOR, L.A.F., HERNANDEZ, M.F.G. **Abordagem sobre a Tecnologia RFID UHF e suas Aplicações.** Perspectivas em Ciências Tecnológicas, vol. 4, 2015, n. 4, p. 135-147.

PISA, P. **O que é e como usar o MySQL?** Disponível em: <https://www.techtudo.com.br/artigos/noticia/2012/04/o-que-e-e-como-usar-o-mysql.html>. Acesso em 28/05/2019.

RFID Journal. **Glossário.** Disponível em: <https://brasil.rfidjournal.com/glossario>. Acesso em 11/06/2019.

SILVA, C.E.A. **Desenvolvimento de Biblioteca para Aplicações de PNRD e PNRD Invertida Embarcadas em Arduino.** Monografia (Graduação em Engenharia Mecatrônica) – Universidade Federal de Uberlândia, Minas Gerais, Brazil, 2017.

SILVEIRA, C.B. **O Que é Indústria 4.0 e Como Ela Vai Impactar o Mundo.** Disponível em: <https://www.citisystems.com.br/industria-4-0/>. Acesso em 28/05/2019.

TAVARES, J.J.P.Z.S., SARAIVA, T.A. **Elementary Petri Nets inside RFID Database (PNRD).** Journal International Journal of Production Research, vol. 48, 2010 – n. 9: RFID Technology and Applications in Production and Supply Chain Management.

TAVARES, J.J.J.P.Z.S., SILVA C.E.A., SOUZA A.R., FONSECA J.P.S. **Monitoramento de Processo da Indústria 4.0 Através da PNRD Integrada com CPN Tools.** Laboratório de

Planejamento Automático da Manufatura – Universidade Federal de Uberlândia, Minas Gerais, Brazil, 2018.

THOMSEN, A. **O que é Arduino?** Disponível em: <https://www.filipeflop.com/blog/o-que-e-arduino/>. Acesso em 28/05/2019.

WILSON, **NFC library for Arduino using PN532.** Disponível em: <https://github.com/elechouse/PN532>. Acesso em 12/04/2019.

APÊNDICES

APÊNDICE A. Código do Arduino.

```
#include <Pn532NfcReader.h>
#include <PN532_HSU.h>
#include <NfcAdapter.h>

//Definição das variáveis utilizadas
int greenpin = 6;
int redpin = 7;
int yellowpin = 8;
int whitepin = 9;
int botao = 10;
int estado;

uint16_t tv[9];
String tag;
String tagID;
String lugar;
String dados;
String conf;
String conf2;
String cond;
String cond2;

byte modo;
byte local;
bool laco;
bool laco2;
bool laco3;

//Definição das informações da etiqueta
int8_t mIncidenceMatrix[] = { -1, -1, -1, 1, 1, 0, 0, 0, 0, 1, 0,
                              1, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0,
                              0, 1, 0, 0, -1, 0, 0, 0, 0, 0, 0,
                              0, 0, 1, 0, 0, -1, 0, 0, 0, 0, 0,
                              0, 0, 0, 0, 0, 1, -1, 0, 0, 0, -1,
                              0, 0, 0, 0, 0, 0, 1, -1, 0, 0, 0,
                              0, 0, 0, 0, 0, 0, 0, 1, -1, 0, 0,
                              0, 0, 0, 0, 0, 0, 0, 0, 1, -1, 0,
                              0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 };

uint16_t mStartingTokenVector[] = { 1, 0, 0, 0, 0, 0, 0, 0, 0 };

//Definição da comunicação com o leitor RFID PN532
PN532_HSU pn532hsu(Serial1);
NfcAdapter nfc = NfcAdapter(pn532hsu);

//Criação dos objetos de leitor e PNRD
Pn532NfcReader* reader = new Pn532NfcReader(&nfc);
Pnrd pnrd = Pnrd(reader, 9, 11);

void setup() {

pinMode(greenpin, OUTPUT);
pinMode(redpin, OUTPUT);
pinMode(yellowpin, OUTPUT);
pinMode(whitepin, OUTPUT);
pinMode(botao, INPUT);

//Iniciação do leitor e da UART
Serial.begin(9600);
```

```
reader->initialize();
nfc.begin();

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

}

void loop() {

delay(1000);

//Aguarda o dado vindo do Python
if (Serial.available() > 0){
  modo = Serial.read();

  //Modo de cadastro
  if (modo == '1'){
    digitalWrite(yellowpin, HIGH);
    laco = true;
    laco3 = true;

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

    while (laco == true){
      //Aguarda a leitura da tag
      if (nfc.tagPresent()){
         NfcTag tag = nfc.read();
         tagID = tag.getUidString();

         while(laco3 == true){
           //Tentativa de escrita na tag
           if (pnrd.saveData() == WriteError::NO_ERROR){
             digitalWrite(greenpin, HIGH);
             delay(1000);
             digitalWrite(greenpin, LOW);
             laco3 = false;
           }
           else {
             digitalWrite(redpin, HIGH);
             delay(1000);
             digitalWrite(redpin, LOW);
           }
         }

         //Condição
         pnrd.getTokenVector(tv);

         //Manda os dados de localização e tagID para o Python
         if(tv[0]==1){
           lugar = "1";
           dados = tagID + "|" + lugar;
           Serial.println(dados);
           laco = false;
         }
      }
    }
    digitalWrite(yellowpin, LOW);
  }

  //Modo de movimentação
  if(modo == '2'){
    digitalWrite(whitepin, HIGH);
    laco = true;
    laco2 = true;
    laco3 = true;
```

```
while (laco == true){

  //Leitura da tag
  ReadError readError = pnrd.getData();

  //Condição
  pnrd.getTokenVector(tv);

  if(readError == ReadError::NO_ERROR){

  //Aguarda a leitura da tag
  if (nfc.tagPresent()){
     NfcTag tag = nfc.read();
     tagID = tag.getUidString();
     digitalWrite(greenpin, HIGH);
     delay(1000);
     digitalWrite(greenpin, LOW);

     if(tv[0]==1){
       //Manda os dados de localização e tagID para o Python
       lugar = "1";
       dados = tagID + "|" + lugar;
       Serial.println(dados);

       while (laco2 == true){

         //Espera a escolha do local de destino pelo usuário
         if (Serial.available() > 0){
           local = Serial.read();

           //Indo para a Oficina
           if (local == '3'){
             delay(2000);

             //Disparo da transição 0
             FireError fireError = pnrd.fire(0);

             switch(fireError){

                case FireError::NO_ERROR:
                  while (laco3 == true){
                    //Salva a nova informação na tag
                    if(pnrd.saveData() == WriteError::NO_ERROR){
                      digitalWrite(greenpin, HIGH);
                      delay(1000);
                      digitalWrite(greenpin, LOW);
                      conf = "ok";
                      conf2 = conf + "|";
                      Serial.println(conf2);
                      laco3 = false;
                    }
                    else {
                      digitalWrite(redpin, HIGH);
                      delay(1000);
                      digitalWrite(redpin, LOW);
                    }
                  }
                  break;

                case FireError::PRODUCE_EXCEPTION:
                  break;

                case FireError::CONDITIONS_ARE_NOT_APPLIED:
                  break;
             }
             laco = false;
             laco2 = false;
```

```
}

//Indo para o Serviço Externo
if (local == '4'){
  delay(2000);

  //Disparo da transição 1
  FireError fireError = pnrd.fire(1);

  switch(fireError){

     case FireError::NO_ERROR:
       while (laco3 == true){
         //Salva a nova informação na tag
         if(pnrd.saveData() == WriteError::NO_ERROR){
           digitalWrite(greenpin, HIGH);
           delay(1000);
           digitalWrite(greenpin, LOW);
           conf = "ok";
           conf2 = conf + "|";
           Serial.println(conf2);
           laco3 = false;
         }
         else {
           digitalWrite(redpin, HIGH);
           delay(1000);
           digitalWrite(redpin, LOW);
         }
       }
       break;

     case FireError::PRODUCE_EXCEPTION:
       break;

     case FireError::CONDITIONS_ARE_NOT_APPLIED:
       break;
  }
  laco = false;
  laco2 = false;
}

//Indo para o Suprimento
if (local == '5'){
  delay(2000);

  //Disparo da transição 2
  FireError fireError = pnrd.fire(2);

  switch(fireError){

     case FireError::NO_ERROR:
       while (laco3 == true){
         //Salva a nova informação na tag
         if(pnrd.saveData() == WriteError::NO_ERROR){
           digitalWrite(greenpin, HIGH);
           delay(1000);
           digitalWrite(greenpin, LOW);
           conf = "ok";
           conf2 = conf + "|";
           Serial.println(conf2);
           laco3 = false;
         }
         else {
           digitalWrite(redpin, HIGH);
           delay(1000);
           digitalWrite(redpin, LOW);
         }
       }
```

```
            break;

          case FireError::PRODUCE_EXCEPTION:
            break;

          case FireError::CONDITIONS_ARE_NOT_APPLIED:
            break;
        }
        laco = false;
        laco2 = false;
      }
    }
  }
}

if(tv[1]==1){
  //Manda os dados de localização e tagID para o Python
  lugar = "2";
  dados = tagID + "|" + lugar;
  Serial.println(dados);

  //Disparo da transição 3
  FireError fireError = pnrd.fire(3);

  switch(fireError){

     case FireError::NO_ERROR:
       while (laco3 == true){
         //Salva a nova informação na tag
         if(pnrd.saveData() == WriteError::NO_ERROR){
           digitalWrite(greenpin, HIGH);
           delay(1000);
           digitalWrite(greenpin, LOW);
           laco3 = false;
         }
         else {
           digitalWrite(redpin, HIGH);
           delay(1000);
           digitalWrite(redpin, LOW);
         }
       }
       break;

     case FireError::PRODUCE_EXCEPTION:
       break;

     case FireError::CONDITIONS_ARE_NOT_APPLIED:
       break;
  }
  laco = false;
}

if(tv[2]==1){
  //Manda os dados de localização e tagID para o Python
  lugar = "3";
  dados = tagID + "|" + lugar;
  Serial.println(dados);

  //Disparo da transição 4
  FireError fireError = pnrd.fire(4);

  switch(fireError){

     case FireError::NO_ERROR:
       while (laco3 == true){
         //Salva a nova informação na tag
         if(pnrd.saveData() == WriteError::NO_ERROR){
           digitalWrite(greenpin, HIGH);
```

```
            delay(1000);
            digitalWrite(greenpin, LOW);
            laco3 = false;
          }
          else {
            digitalWrite(redpin, HIGH);
            delay(1000);
            digitalWrite(redpin, LOW);
          }
        }
        break;

      case FireError::PRODUCE_EXCEPTION:
        break;

      case FireError::CONDITIONS_ARE_NOT_APPLIED:
        break;
    }
    laco = false;
}

if(tv[3]==1){
    //Manda os dados de localização e tagID para o Python
    lugar = "4";
    dados = tagID + "|" + lugar;
    Serial.println(dados);

    //Disparo da transição 5
    FireError fireError = pnrd.fire(5);

    switch(fireError){

      case FireError::NO_ERROR:
        while (laco3 == true){
          //Salva a nova informação na tag
          if(pnrd.saveData() == WriteError::NO_ERROR){
            digitalWrite(greenpin, HIGH);
            delay(1000);
            digitalWrite(greenpin, LOW);
            laco3 = false;
          }
          else {
            digitalWrite(redpin, HIGH);
            delay(1000);
            digitalWrite(redpin, LOW);
          }
        }
        break;

      case FireError::PRODUCE_EXCEPTION:
        break;

      case FireError::CONDITIONS_ARE_NOT_APPLIED:
        break;
    }
    laco = false;
}

if(tv[4]==1){
    estado = digitalRead(botao);

    //Manda os dados de localização e tagID para o Python
    lugar = "5";
    dados = tagID + "|" + lugar;
    Serial.println(dados);

    delay(3000);
    if (estado == HIGH){
```

```
  digitalWrite(yellowpin, HIGH);

  //Disparo da transição 10
  FireError fireError = pnrd.fire(10);

  switch(fireError){

     case FireError::NO_ERROR:
       while (laco3 == true){
         //Salva a nova informação na tag
         if(pnrd.saveData() == WriteError::NO_ERROR){
           digitalWrite(greenpin, HIGH);
           delay(1000);
           digitalWrite(greenpin, LOW);
           cond = "quar";
           cond2 = cond + "|";
           Serial.println(cond2);
           laco3 = false;
         }
         else {
           digitalWrite(redpin, HIGH);
           delay(1000);
           digitalWrite(redpin, LOW);
         }
       }
       break;

     case FireError::PRODUCE_EXCEPTION:
       break;

     case FireError::CONDITIONS_ARE_NOT_APPLIED:
       break;
  }
  digitalWrite(yellowpin, LOW);
  laco = false;
}

else{
  //Disparo da transição 6
  FireError fireError = pnrd.fire(6);

  switch(fireError){

     case FireError::NO_ERROR:
       while (laco3 == true){
         //Salva a nova informação na tag
         if(pnrd.saveData() == WriteError::NO_ERROR){
           digitalWrite(greenpin, HIGH);
           delay(1000);
           digitalWrite(greenpin, LOW);
           cond = "cali";
           cond2 = cond + "|";
           Serial.println(cond2);
           laco3 = false;
         }
         else {
           digitalWrite(redpin, HIGH);
           delay(1000);
           digitalWrite(redpin, LOW);
         }
       }
       break;

     case FireError::PRODUCE_EXCEPTION:
       break;

     case FireError::CONDITIONS_ARE_NOT_APPLIED:
       break;
```

```
    }
    laco = false;
  }
}

if(tv[5]==1){
  //Manda os dados de localização e tagID para o Python
  lugar = "6";
  dados = tagID + "|" + lugar;
  Serial.println(dados);

  //Disparo da transição 7
  FireError fireError = pnrd.fire(7);

  switch(fireError){

     case FireError::NO_ERROR:
       while (laco3 == true){
         //Salva a nova informação na tag
         if(pnrd.saveData() == WriteError::NO_ERROR){
           digitalWrite(greenpin, HIGH);
           delay(1000);
           digitalWrite(greenpin, LOW);
           laco3 = false;
         }
         else {
           digitalWrite(redpin, HIGH);
           delay(1000);
           digitalWrite(redpin, LOW);
         }
       }
       break;

     case FireError::PRODUCE_EXCEPTION:
       break;

     case FireError::CONDITIONS_ARE_NOT_APPLIED:
       break;
  }
  laco = false;
}

if(tv[6]==1){
  //Manda os dados de localização e tagID para o Python
  lugar = "5";
  dados = tagID + "|" + lugar;
  Serial.println(dados);

  //Disparo da transição 8
  FireError fireError = pnrd.fire(8);

  switch(fireError){

     case FireError::NO_ERROR:
       while (laco3 == true){
         //Salva a nova informação na tag
         if(pnrd.saveData() == WriteError::NO_ERROR){
           digitalWrite(greenpin, HIGH);
           delay(1000);
           digitalWrite(greenpin, LOW);
           laco3 = false;
         }
         else {
           digitalWrite(redpin, HIGH);
           delay(1000);
           digitalWrite(redpin, LOW);
         }
       }
```

```
            break;

          case FireError::PRODUCE_EXCEPTION:
            break;

          case FireError::CONDITIONS_ARE_NOT_APPLIED:
            break;
        }
        laco = false;
      }

      if(tv[7]==1){
        //Manda os dados de localização e tagID para o Python
        lugar = "4";
        dados = tagID + "|" + lugar;
        Serial.println(dados);

        //Disparo da transição 9
        FireError fireError = pnrd.fire(9);

        switch(fireError){

          case FireError::NO_ERROR:
            while (laco3 == true){
              //Salva a nova informação na tag
              if(pnrd.saveData() == WriteError::NO_ERROR){
                digitalWrite(greenpin, HIGH);
                delay(1000);
                digitalWrite(greenpin, LOW);
                laco3 = false;
              }
              else {
                digitalWrite(redpin, HIGH);
                delay(1000);
                digitalWrite(redpin, LOW);
              }
            }
            break;

          case FireError::PRODUCE_EXCEPTION:
            break;

          case FireError::CONDITIONS_ARE_NOT_APPLIED:
            break;
        }
        laco = false;
      }

      if(tv[8]==1){
        //Manda os dados de localização e tagID para o Python
        lugar = "7";
        dados = tagID + "|" + lugar;
        Serial.println(dados);

        digitalWrite(redpin, HIGH);
        delay(1000);
        digitalWrite(redpin, LOW);
        laco = false;
        modo = ' ';
      }
    }
  }
  }
  digitalWrite(whitepin, LOW);
 }
}
}
```

APÊNDICE B. Código do Python.

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-
import time
import serial
comport = serial.Serial('COM5', 9600)
print ('Serial Iniciada...\n')

import mysql.connector
cnx = mysql.connector.connect(user='root', password='root', host='127.0.0.1',
database='ferramentaria')
cursor = cnx.cursor()

while (True):
    fo = open("comunicacao.txt", "r")
    tabela = fo.read()
    fo.close()

    if tabela != '':
        if tabela == "fer":
            time.sleep(1)
            comport.write(b'1')

            serialValue = comport.readline()
            data_dados = serialValue.split(b"|")
            print(data_dados)
            cursor.execute("INSERT INTO ferramentas (tagID, localizacao) VALUES
(%s, %s)", data_dados)

        if tabela == "mov":
            time.sleep(1)
            comport.write(b'2')

            serialValue = comport.readline()
            data_dados = serialValue.split(b"|")
            print(data_dados)
            cursor.execute("INSERT INTO movimentacao (tagID, localizacao) VALUES
(%s, %s)", data_dados)

        fo = open("comunicacao.txt", "w")
        fo.close()

    fo2 = open("comunicacao2.txt", "r")
    local = fo2.read()
    fo2.close()

    if local != '':
        if local == "2":
            time.sleep(1)
            comport.write(b'3')

            confirmacao = comport.readline()
            (confirmacao2, confirmacao3) = confirmacao.split(b"|")

            encoding = 'utf-8'
            conf = str(confirmacao2, encoding)

            if conf == "ok":
                fo3 = open("confirmacao.txt", "w")
                fo3.write("1")
                fo3.close()

        if local == "3":
            time.sleep(1)
            comport.write(b'4')
```

```
            confirmacao = comport.readline()
            (confirmacao2, confirmacao3) = confirmacao.split(b"|")

            encoding = 'utf-8'
            conf = str(confirmacao2, encoding)

            if conf == "ok":
                fo3 = open("confirmacao.txt", "w")
                fo3.write("1")
                fo3.close()

        if local == "4":
            time.sleep(1)
            comport.write(b'5')

            confirmacao = comport.readline()
            (confirmacao2, confirmacao3) = confirmacao.split(b"|")

            encoding = 'utf-8'
            conf = str(confirmacao2, encoding)

            if conf == "ok":
                fo3 = open("confirmacao.txt", "w")
                fo3.write("1")
                fo3.close()

        fo2 = open("comunicacao2.txt", "w")
        fo2.close()

    fo4 = open("condicao.txt", "r")
    condicao = fo4.read()
    fo4.close()

    if condicao != '':
        if condicao == "quar":
            cond = comport.readline()
            (cond2, cond3) = cond.split(b"|")

            encoding = 'utf-8'
            condi = str(cond2, encoding)

            if condi == "quar":
                fo5 = open("condicao2.txt", "w")
                fo5.write("1")
                fo5.close()

            if condi == "cali":
                fo5 = open("condicao2.txt", "w")
                fo5.write("2")
                fo5.close()

        fo4 = open("condicao.txt", "w")
        fo4.close()

    cnx.commit()

cursor.close()
cnx.close()
comport.close()
```

APÊNDICE C. Plano de Trabalho.

PLANO DE TRABALHO DO PROJETO FINAL

Título: Manufatura digital: uma contribuição da tecnologia PNRD aplicada ao controle de inventários

Dados	
Aluno:	Pollyana Alves Resende
Nº de Matrícula:	201301606
Telefones:	(62) 98452-3089
E-mail:	pollyanaalvesresende@gmail.com
Orientador:	João Paulo da Silva Fonseca
Curso:	Eng. Elétrica ( ) Eng. de Computação ( ) Eng. Mecânica (X)
Certif. Estudos	Não (-) Sim (-)

Resumo

O conceito de Industria 4.0, manufatura avançada ou, ainda, digitalização da manufatura, aponta para a digitalização dos processos produtivos, interconexão entre os dispositivos de campo, coleta e análise de dados, diagnósticos e tomadas de decisão de forma autônoma. Entretanto, ambientes industriais atuais carecem de maiores avanços nos recursos de infraestrutura de comunicação, evolução de técnicas de inteligência artificial e segurança de dados digitais. Em um ambiente no qual os produtos assumem funções ativas no processo produtivo, tornam-se essenciais ferramentas que proporcionem a comunicação entre os demais dispositivos de campo em um ambiente integrado de comunicação. Este projeto final consiste na aplicação direta de técnica computacional em dispositivos de campo, através da rede de Petri inserida em base de dados RFID, ou *Petri net inside RFID database* (PNRD), aplicada em tarefas de controle de inventário e ciclo de vida de produtos. Espera-se que a tecnologia PNRD garanta maior autonomia ao processo produtivo, melhorando a rastreabilidade de produtos e aumentando o controle do ciclo de vida de ativos com a digitalização da manufatura.

I. Objetivos

Os objetivos do presente projeto são:

- Especificar o(s) itens(s) objeto de estudo;
- Modelar o comportamento do(s) itens(s) em rede de Petri lugar/transição;
- Adaptar dispositivo embarcado associado ao(s) itens(s) objeto de estudo;
- Implementar modelo comportamental diretamente em dispositivos de campo;
- Preparar pelo menos três cenários de ensaio;
- Realizar testes e ensaios;
- Coletar e analisar os resultados;
- Documentar o material em forma de monografia para disponibilização pública.

II. Metodologia

A metodologia utilizada neste projeto será:

- Identificação da demanda por digitalização da rastreabilidade do ciclo de vida de determinado(s) itens(s) no processo produtivo;
- Modelagem do comportamento do(s) itens(s) objeto de estudo;

- Integração do(s) itens(s) com dispositivo embarcado para viabilizar o controle discreto;
- Implementação do modelo comportamental nos dispositivos de campo;
- Definição de cenários e realização de testes e ensaios;
- Documentação dos resultados na forma de uma monografia de conclusão de curso.

III. Cronograma

O cronograma das atividades a serem realizadas é apresentado na Tabela 1.

Tabela 1 - Cronograma do projeto 1º Semestre letivo de 2019.

Etapas do Projeto	MAR	ABR	MAI	JUN	JUL
1) Levantamento de requisitos	X				
2) Especificação de materiais	X	X			
3) Modelagem do ciclo de vida dos objetos	X	X	X		
4) Integração com dispositivos embarcados		X	X	X	
5) Realização de testes, ensaios e ajustes			X	X	
6) Redação da monografia			X	X	
7) Defesa do projeto final de curso					X

Goiânia, 19 de março de 2019.

Pollyana Alves Resende
Matrícula: 201301606

João Paulo da Silva Fonseca
Prof. Orientador