



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

ALAN ROGÉRIO BARRETO PIRES

**SISTEMA CIBER-FÍSICO BASEADO NA INTEGRAÇÃO
PNRD E IPNRD UTILIZANDO CLOUD COMPUTING**

**UBERLÂNDIA
2019**

ALAN ROGÉRIO BARRETO PIRES

**SISTEMA CIBER-FÍSICO BASEADO NA INTEGRAÇÃO
PNRD E IPNRD UTILIZANDO CLOUD COMPUTING**

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

Área de concentração: Engenharia Mecatrônica

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

**UBERLÂNDIA
2019**

*Este trabalho é dedicado à minha família, amigos e demais pessoas que passaram pela
minha vida e me tornaram quem eu sou hoje.*

AGRADECIMENTOS

À Deus que permitiu e proporcionou todas as condições para que eu chegasse aqui.

Aos meus pais que sempre me apoiaram e sempre se mantiveram presentes me ajudando sempre que precisei e me dando todo tipo de suporte, desde as conversas de madrugada pra desabafar, até compra de passagens para que eu pudesse visita-los.

Ao professor Dr. José Jean-Paul Zanlucchi de Souza Tavares, por não só me orientar, como também inspirar, e me incentivar a sempre fazer um trabalho melhor para que minha contribuição pudesse ser valorizada não só pelos outros como por mim mesmo.

Ao Gabriel de Almeida Souza que fez os códigos que usei como base para o desenvolvimento desse trabalho, e que se mostrou várias vezes disposto a me ajudar quando tive dificuldades.

Aos demais membros do MAPL, Thiago, Marcos, Leandro, Nina, entre outros que sempre se mostraram dispostos a tirar dúvidas e ajudar, não só na parte técnica mas também mantendo um ambiente bem humorado e produtivo dentro do laboratório.

Aos meus colegas de turma e amigos que obtive durante a graduação e me ajudaram a atravessar este período de forma agradável.

Aos professores que tive, que foram essenciais para minha formação.

“Saber muito não lhe torna inteligente. A inteligência se traduz na forma que você recolhe, julga, maneja e, sobretudo, onde e como aplica essa informação”
Carl Sagan

RESUMO

Os crescentes avanços tecnológicos da indústria estão convergindo para a aquisição e utilização inteligente de dados. A quarta revolução industrial introduziu diversos conceitos neste contexto. Atualmente, vem-se utilizando da Internet das Coisas para a integração de sistemas ciber-físicos e para se reunir e utilizar massivas quantidades de dados para a personalização e melhora dos meios de produção. No que concerne sistemas ciber-físicos, os equipamentos passam a ter acesso a servidores mais potentes por meio da conexão *Ethernet*, que os oferece grande poder computacional e assim otimiza o desempenho do sistema como um todo. Este trabalho se baseia nestes paradigmas e faz a utilização de PNRD e *iPNRD* aplicado em *Cloud Computing* para o planejamento automático de processos que envolvam sistemas a eventos discretos. Será utilizado o problema clássico do Mundo de Blocos como exemplo, onde um braço robótico equipado a um leitor RFID irá identificar o arranjo dos blocos e movê-los para um arranjo desejado, e então, ao fim, será feita uma análise a respeito do desempenho deste projeto em comparação com outro implementado utilizando *Edge Computing*, onde será constatado em quais situações é mais vantajoso a utilização do sistema em *Cloud* e quando é mais seguro se operar com *Edge*.

Palavras-chave: Sistemas Ciber-Físicos, Planejamento de processos, PNRD/*iPNRD*, Cloud Computing.

ABSTRACT

Industry's increasing technological advances are converging on intelligent data acquisition and utilization. The fourth industrial revolution introduced several concepts in this context. Nowadays, the Internet of Things is being used to integrate cyber-physical systems and to gather and use massive amounts of data to customize and improve the means of production. As far as cyber-physical systems are concerned, equipment now has access to more powerful servers via the Ethernet connection, which provides them with high computational power and thus optimizes overall system performance. This work approaches the use of PNRD and iPNRD applied in Cloud Computing for the automatic planning of processes involving discrete event systems. The Block World classic problem will be used as an example, where a robotic arm equipped with a RFID reader will identify the arrangement of blocks and then, move them to a desired position and then, at the end, an analysis will be made of the performance of this project compared to another implemented using Edge Computing, where will be noted in which situations it is most advantageous to use the Cloud system and when it is safer to operate with Edge one.

Key-words: Cyber-Physical Systems, Process Planning, PNRD/iPNRD, Cloud-Computing.

LISTA DE ILUSTRAÇÕES

Figura 1 – Componentes de um sistema RFID	15
Figura 2 – Elementos Básicos de uma rede de Petri	17
Figura 3 – Rede de Petri representando os momentos do dia	17
Figura 4 – Exemplo de rede de Petri representando o funcionamento de um semáforo	18
Figura 5 – Representação simplificada de um processo de verificação de mancais usando redes de Petri	21
Figura 6 – Associação dos termos da Equação 2.1 com os elementos onde serão armazenados nas abordagens PNRD e <i>i</i> PNRD.	22
Figura 7 – Matriz representando o espaço de trabalho discretizado.	23
Figura 8 – Sistema Ciber-Físico para solução do Mundo de Blocos utilizando a integração de PNRD e <i>i</i> PNRD	24
Figura 9 – Modelo <i>i</i> PNRD	25
Figura 10 – Espaço de Petri	26
Figura 11 – Modelo PNRD (perspectiva do bloco)	27
Figura 12 – Planejamento de processos a eventos discretos utilizando rede de Petri em Cloud Computing	28
Figura 13 – Braço robótico equipado com leitor RFID ao lado dos blocos etiquetados	29
Figura 14 – Arduino Mega acoplado ao módulo Ethernet W5100	30
Figura 15 – Fluxograma de ações do sistema local	31
Figura 16 – Planejamento ideal	31
Figura 17 – Planejamento real	33
Figura 18 – Mensagem com o planejamento de ações	34

LISTA DE ABREVIATURAS E SIGLAS

PNRD	Base de dados inseridas em RFID
iPNRD	PNRD invertida
RFID	Identificação por Rádio-Frequência
ID	Identificação
IoT	Internet das coisas
IIoT	Internet das coisas industrial
PN	Rede de Petri
CPU	Unidade Central de Processamento
GB	Gigabyte
GHz	Giga-Hertz
NFC	Comunicação por Campo de Proximidade
RAM	Memória de Acesso Aleatório
HTTP	Protocolo de Transferência de Hipertexto
PHP	Páginas de Hipertexto Pré-processadas
IDE	Ambiente de Desenvolvimento Integrado
URL	Localizador Universal de Recursos
IP	Protocolo de Internet
MQTT	<i>Message Queuing Telemetry Transport</i>

LISTA DE SÍMBOLOS

P	Número finito de estados
T	Conjunto de transições
A	Conjunto de arcos direcionados
W	Função peso
M_K	Vetor de marcações
A^T	Matriz de adjacência
u_k	Vetor de disparos
T_k	Transição k, onde $k = 0, 1, 2, \dots, n$
$\&$	Símbolo grego E
T_p	Tempo de planejamento
T_c	Tempo de estabelecimento de comunicação
T_m	Tempo de troca de mensagens
E	Estado
ms	Milissegundos
kB	Quilobytes
GB	Gigabytes
$\%$	Porcento

SUMÁRIO

1	INTRODUÇÃO	11
1.1	OBJETIVOS	12
1.1.1	Objetivo principal	12
1.1.2	Objetivos específicos	12
1.2	JUSTIFICATIVA	13
2	FUNDAMENTAÇÃO TEÓRICA	14
2.1	AUTO-ID	14
2.1.1	Radio Frequency Identification (RFID)	14
2.2	INTERNET DAS COISAS	15
2.2.1	Internet Industrial	16
2.3	REDES DE PETRI	16
2.3.1	Representação Gráfica	16
2.3.2	Representação matemática	17
2.4	REDE DE PETRI INSERIDA EM BASE DE DADOS RFID (PNRD)	19
2.5	PNRD INVERTIDA (IPNRD)	21
2.6	ESTUDO DE CASO: MUNDO DE BLOCOS	22
3	METODOLOGIA E DESENVOLVIMENTO	24
3.1	ESPECIFICAÇÃO DA SOLUÇÃO	24
3.1.1	Sistema em nuvem	25
3.1.1.1	Exemplo	28
3.1.2	Sistema local	28
3.2	TESTES	30
4	RESULTADOS E DISCUSSÃO	33
5	CONCLUSÃO	36
	REFERÊNCIAS	37
	APÊNDICE A – PROGRAMA DO SERVIDOR WEB	39
	APÊNDICE B – PROGRAMA DO SISTEMA LOCAL	47
	APÊNDICE C – ELEMENTOS DO MODELO IPNRD	68
	APÊNDICE D – ELEMENTOS DO MODELO PNRD	70
	APÊNDICE E – DESCRIÇÃO DAS POSIÇÕES REFEREN- TES A CADA LETRA DA MENSAGEM ENVIADA AO SISTEMA LOCAL	71

1 INTRODUÇÃO

As indústrias são as partes da economia responsáveis pela produção dos bens. Geralmente, são altamente mecanizadas e automatizadas. Desde o início da industrialização, saltos tecnológicos levaram a grandes mudanças de paradigma, estes momentos são denominados, atualmente, como "revoluções industriais": no campo da mecanização (primeira revolução industrial), do uso intensivo de energia elétrica (segunda revolução), da digitalização generalizada (terceira revolução), e nos dias atuais, é perceptível o surgimento de um novo paradigma, marcado pela integração da Internet com as tecnologias associadas a objetos inteligentes e pela computação em nuvem. Nele a produção contém sistemas modulares e eficientes, nos quais os produtos controlam seu próprio processo de fabricação. Isto acarretando em produtos individuais, totalmente personalizados, feitos em lotes de uma unidade, porém mantendo as condições econômicas da produção em massa. Essa é a perspectiva da quarta revolução industrial ou, como também é chamada, Indústria 4.0. (LASI et al., 2014)

A quarta revolução industrial introduz diversos conceitos que servem como base para se atingir seus objetivos, dentre estes conceitos estão a Internet das Coisas, robôs autônomos, integração de sistemas, computação em nuvem, *Big Data*, entre diversos outros que estão sendo aplicados na indústria para atingir os fins supracitados.

Um conceito de grande relevância é o da Internet das Coisas, que proporciona conectividade, não só para qualquer um, mas para qualquer coisa. Se trata de um mundo onde objetos físicos, dados virtuais, meio ambiente e os seres humanos podem estar conectados e podem interagir um com o outro de qualquer lugar e a qualquer momento. (SUNDMAEKER et al., 2010).

A computação em nuvem é a evolução natural da convergência de muitas tecnologias e conceitos. Ela descreve um ambiente de computação baseada na gigantesca rede de servidores, sendo estes, virtuais ou físicos (TAURION, 2009). De acordo com Mell, Grance et al. (2011), este modelo vai proporcionar acesso a uma rede onipresente, conveniente e sob demanda de diversos recursos computacionais configuráveis. Uma característica importante deste modelo é chamada *On-Demand self-service*, que significa self-service sob demanda, onde um servidor provém capacidades computacionais como armazenamento em rede ou poder de processamento e um consumidor qualquer pode solicitar de qualquer lugar e a qualquer momento.

Nos últimos anos, com o rápido aumento do uso de dispositivos finais conectados a Internet, o que inclui tecnologias RFID, este paradigma centralizado muitas vezes não pôde atender aos rigorosos requisitos em tempo real de muitas aplicações devido as latências de

comunicação (LIU et al., 2018). Além disto, falhas de conexão podem acabar interrompendo os processos e parando a produção. A tecnologia RFID, que se trata de um sistema de identificação automática, pode auxiliar o sistema localmente como uma abordagem em Edge Computing, e isto pode funcionar como um sistema complementar e/ou contingencial para o sistema na nuvem em caso de falha na rede. Nesta mesma perspectiva, a abordagem PNRD, que consiste na inserção de informações de uma rede de Petri em um sistema RFID, permite que o controle de fluxo do processo seja feito na fronteira, o que reduz a possibilidade de falhas por conta de comunicação.

A PNRD e a iPNRD são ótimas ferramentas de análise de processos, levando o controle dos mesmos para as fronteiras, onde falhas podem ser identificadas imediatamente por leitores RFID. Além disto, pela simplicidade das redes de Petri é possível que sistemas complexos de manufatura possam ser modelados de forma prática e simples.

Este trabalho visa o planejamento automático de soluções para o problema do Mundo de Blocos utilizando computação em nuvem, e aplicando diretamente o conceito de Internet das Coisas, auxiliada pela abordagem iPNRD.

1.1 OBJETIVOS

1.1.1 Objetivo principal

O objetivo principal é desenvolver um sistema ciber-físico baseado na integração PNRD e iPNRD utilizando *cloud computing* para solução do problema do Mundo de Blocos.

1.1.2 Objetivos específicos

- Projetar um servidor WEB;
- Projetar um sistema local para controle de um braço robótico;
- Estabelecer comunicação entre o servidor WEB e o braço robótico;
- Acoplar um leitor RFID ao braço robótico;
- Estabelecer comunicação do braço robótico com o leitor RFID;
- Fazer a integração dos sistemas envolvidos;
- Projetar um programa para o sistema local onde ele deve ser responsável por identificar os blocos e como estão arrançados e fazer solicitações de planejamento de ações para o sistema em nuvem, e após isto, fazer as movimentações necessárias.

- Projetar um sistema em nuvem capaz de fazer o planejamento necessário com base nas informações cedidas pelo sistema local.

1.2 JUSTIFICATIVA

O rápido avanço tecnológico desenvolvido nas indústrias e no meio acadêmico nos trouxe a um patamar onde os processos produtivos são altamente voláteis. Em curtos períodos de tempo, as indústrias precisam mudar seus métodos de manufatura para se manterem competitivas no mercado. Neste contexto surgiram as células flexíveis de manufatura, porém elas não são eficazes sem um ótimo planejamento de processos. As técnicas de planejamento, modelagem e controle de processos ainda são dispendiosas, e é neste sentido que este trabalho opera.

Ao se projetar um servidor WEB capaz de entregar um novo planejamento de fluxo sempre que solicitado, se comunicando diretamente com os microcontroladores nas fronteiras da rede, proporciona-se uma maior fluidez em todos os processos, além de um rápido tratamento de erros, aumentando a produtividade, reduzindo custos e, portanto, aumentando o lucro das empresas.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, são tratados diversos assuntos, que, a princípio, podem parecer desconexos, porém são de fundamental importância para o desenvolvimento e entendimento do trabalho. Primeiramente são abordados os sistemas de identificação, após isto, é introduzido o conceito de Internet das Coisas, que é a base deste trabalho. Após, são explicadas as redes de Petri, e sua aplicação utilizando PNRD e *i*PNRD e então, um breve estudo sobre a problemática do trabalho, o Mundo de Blocos.

2.1 AUTO-ID

Atualmente, os procedimentos de identificação automática vem se tornando comuns em vários ramos da indústria, como compra, logística, manufatura, fluxo de materiais, etc. O sistema de identificação automática existe para prover informações acerca de objetos de interesse, previamente etiquetados, podendo estes ser pessoas (cartões de acesso), animais (coleira de animais de estimação), peças ou produtos em geral.

O código de barras se manteve por mais de 20 anos no topo dos sistemas utilizados para identificação. Por ser extremamente barato e de fácil utilização, ele estava incluído nos mais diversos elementos das cadeias de suprimento. Porém, encontrava limites na sua capacidade de armazenamento e no fato de que ele não pode ser "reprogramado". (FINKENZELLER, 2010)

De acordo com Finkenzeller (2010) e Brock (2001), o avanço da Internet, da digitalização da informação e a globalização dos negócios ofereceram novas possibilidades, quando se trata de identificação e localização de produtos. Para se aproveitar melhor dessas possibilidades, se fez necessária a utilização de um novo sistema de identificação. Um que conferisse ao objeto identificação única, facilitasse a localização deste no seu ciclo de vida, tivesse uma memória maior para a adição de informações adicionais à identificação e que também fosse reprogramável.

2.1.1 Radio Frequency Identification (RFID)

Finkenzeller (2010) afirma que no sistema RFID, os dados são armazenados em um dispositivo eletrônico de memória —o transponder. Uma característica deste dispositivo é que toda a troca de informações entre ele e o leitor é efetuada utilizando campos magnéticos ou eletromagnéticos, o que permite que não haja nenhum tipo de contato entre as partes deste sistema.

Um sistema RFID é composto de dois componentes:

- Transponder (*tag* ou etiqueta): Se localiza no objeto de interesse. É a parte que armazena dados do sistema, contém um microchip eletrônico e uma antena. Quando possui bateria, pode-se dizer que é uma etiqueta ativa, e quando não possui, é passiva.
- Leitor: Dispositivo que pode ser do tipo leitor ou escritor/leitor, dependendo do modelo. É sempre referido como "leitor", independente se ele puder fazer escrita também. Contém um módulo de rádio-frequência, uma unidade de controle e uma antena, a qual permite receber e mandar dados para o transponder.

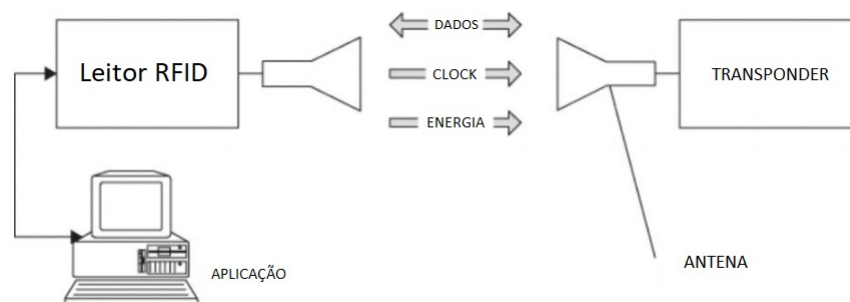


Figura 1 – Componentes de um sistema RFID

Fonte: Finkenzeller (2010)

A tecnologia RFID abre portas para fazer com que organizações e companhias sejam mais seguras, confiáveis e precisas em seus procedimentos. Aplicações como controle de acesso, análise de inventário, processos de produção e manufatura irão possuir uma nova gama de possibilidades e sofrer muitas melhorias com o uso deste sistema de identificação. (AHSAN; SHAH; KINGSTON, 2010)

O RFID, somado as tecnologias de sensores, permitem que computadores possam observar, identificar e entender o mundo – sem as limitações dos seres humanos. (ASHTON et al., 2009)

2.2 INTERNET DAS COISAS

O termo "*Internet of Things*" (IoT) foi primeiramente usado por Kevin Ashton em uma apresentação na Procter & Gamble (P&G) em 1999. O conceito veio da prerrogativa de que a Internet, até então, era somente alimentada por humanos, os quais têm tempo, atenção e precisão limitados para adquirir dados a respeito de coisas do mundo real.

Partindo deste princípio, se os computadores pudessem se comunicar diretamente com as "coisas", eles teriam mais informações e portanto seriam mais precisos para entregar soluções para problemas do mundo físico. (ASHTON et al., 2009)

2.2.1 Internet Industrial

Também conhecida como *Industrial Internet of Things (IIoT)*, ou como denominado pela Cisco: *Internet of Everything* (Internet de Tudo). A Internet Industrial possibilitou uma maneira de se obter mais informações acerca das operações de uma companhia e de seus recursos através de uma integração de sensores, máquinas, *middleware*, *softwares*, sistemas de armazenamento e dos sistemas *back-end* de computação em nuvem.

Apesar de parecida com as tecnologias e técnicas existentes para comunicação máquina-máquina, a *IIoT* abrange proporções muito maiores. Utilizando *Big Data*, imensas quantidades de dados podem ser analisadas em um sistema na nuvem entregando informações e dados estatísticos que jamais poderiam ser obtidos pelos métodos anteriores, em que se dependia do intermédio humano para a alimentação dos bancos de dados. Engenheiros e administradores podem, assim, otimizar os meios de produção e também os modelos de negócio das empresas, trazendo inúmeras possibilidades de melhorias para as indústrias do mundo moderno. Este conceito vem trazendo diversas tecnologias que tem como objetivo criar um sistema maior que a soma de suas partes. (GILCHRIST, 2016)

2.3 REDES DE PETRI

De acordo com Maciel, Lins e Cunha (1996), redes de Petri é uma técnica de especificação de sistemas que permite uma representação gráfica e matemática e possui mecanismos que permitem a verificação tanto das propriedades de um processo como o andamento do mesmo. A partir da utilização de redes de Petri pode-se modelar os mais diversos tipos de sistema, podendo ser compostos por eventos concorrentes, assíncronos, distribuídos, paralelos ou não-determinísticos.

2.3.1 Representação Gráfica

A representação gráfica das redes de Petri permite a visualização dos processos utilizando alguns elementos visuais.

Nesta representação, as redes de Petri possuem dois componentes: as *transições* e os *lugares*. Os lugares correspondem às variáveis de estado e as transições representam as ações ou eventos que acontecem no sistema. A realização de uma ação depende do cumprimento de pré-requisitos do processo. Gráficamente os lugares são representados por círculos e as transições por traços ou barras conforme representado na Figura 2.

Adicionalmente, fichas são usadas nas redes para simular a dinâmica das atividades, ou seja, o evento que está ocorrendo em determinado momento, ou em qual etapa o processo se encontra.

Segundo Murata (1989), as redes de Petri são grafos orientados, onde seus nós são

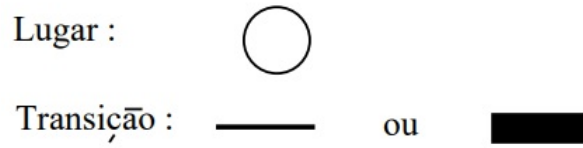


Figura 2 – Elementos Básicos de uma rede de Petri

Fonte: Maciel, Lins e Cunha (1996)

os lugares e transições. Nelas, os arcos podem ser de um lugar para uma transição ou de uma transição para um lugar e nunca diferente disto.

A seguir, na Figura 3, tem-se a representação dos períodos do dia através de uma rede de Petri. Neste modelo, tem-se como exemplo o lugar *manhã* que com a ativação da transição *entardecer*, a ficha muda para o lugar *tarde*.

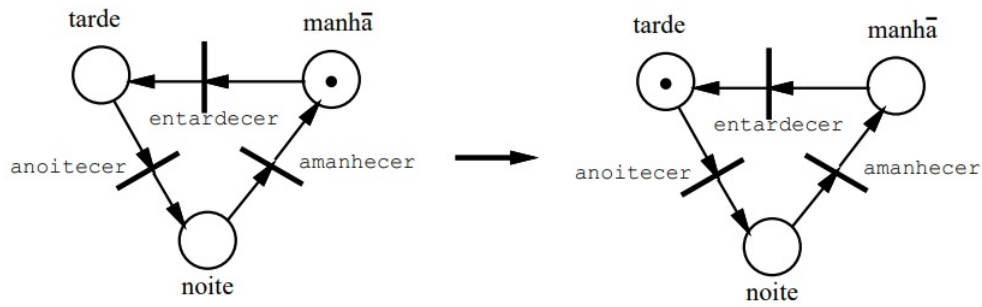


Figura 3 – Rede de Petri representando os momentos do dia

Fonte: Maciel, Lins e Cunha (1996)

2.3.2 Representação matemática

Definição 2.1. Uma rede de Petri pode ser definida como uma tupla de 5 elementos $PN = (P, T, A, W, M_0)$, onde:

$P = \{p_1, p_2, p_3, \dots, p_m\}$, é o número finito de estados.

$T = \{t_1, t_2, t_3, \dots, t_n\}$, é o número finito de transições.

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

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

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

Uma vez definida a rede de Petri, um disparo pode ser caracterizado pela Equação 2.1:

$$M_{k+1} = M_k + A^T \times u_k \quad (2.1)$$

Onde:

M_k é o vetor coluna que possui tantos elementos quanto existem lugares na rede. É denominado como vetor de marcações.

M_{k+1} é o vetor de marcações resultante do disparo.

A^T trata-se da matriz de incidência, ela representa os arcos de uma rede de Petri. O elemento a_{ij} desta matriz é igual a -1 quando a transição j se encontra logo após ao lugar i , ele vale 1 quando a transição j se encontra antes do lugar i , e quando a transição j não acompanha o lugar i , este elemento tem valor 0.

O vetor u_k possui tantos elementos quanto a rede possuir transições. Denominado vetor de disparos, ele possui todos os elementos iguais a zero, exceto o elemento a_k que vale 1.

Para exemplificar, utilizaremos o exemplo de um semáforo, conforme representado na Figura 4.

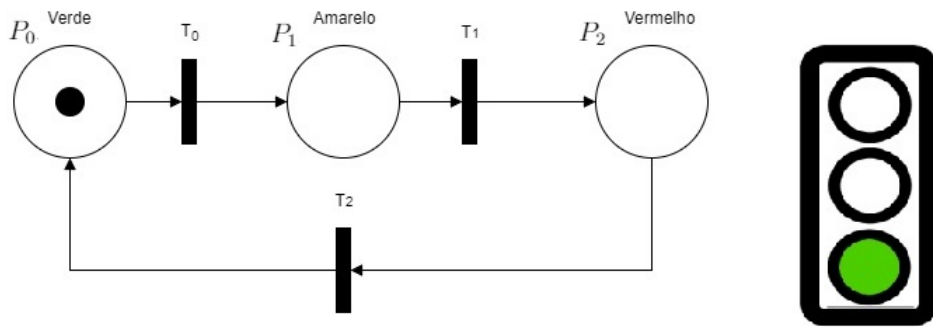


Figura 4 – Exemplo de rede de Petri representando o funcionamento de um semáforo

Fonte: Silva et al. (2017)

Neste exemplo tem-se três lugares e três transições. Segundo as informações supracitadas, tem-se que:

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

Além disto:

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

Neste exemplo, pretendemos ativar a transição T_0 , para que o semáforo vá para o estado amarelo. Então utilizaremos o vetor de disparo u_0 , que corresponde a transição desejada. Portanto:

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

Aplicando estes valores na Equação 2.1, temos:

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

Perceba que este exemplo se assemelha ao apresentado na Figura 3, se tratando somente de um sistema diferente, porém sua rede de Petri é idêntica, podendo ser tratada da mesma maneira.

2.4 REDE DE PETRI INSERIDA EM BASE DE DADOS RFID (PNRD)

De acordo com Aalst (1998) e Tavares e Saraiva (2010), redes de Petri são ótimas ferramentas para modelagem e análise de processos. Por um lado, as redes de Petri podem ser usadas como uma linguagem gráfica para a especificação de fluxos de trabalho complexos, e por outro, a teoria das redes de Petri fornecem poderosas técnicas de análise que podem ser usadas para identificar as falhas dos procedimentos de fluxos de trabalho. Inserir-las em base de dados RFID é o primeiro passo para melhorar a integração de sistemas de controle com a tecnologia RFID.

Tavares e Saraiva (2010) afirmam que do ponto de vista do objeto (etiqueta), ele é apenas uma ficha dentro da rede de Petri que representa o modelo de um processo no qual

este objeto está inserido. Todo o planejamento de processo pode ser armazenado em uma etiqueta RFID na forma de uma matriz de incidência, ela representa o plano de processo. Até mesmo processos flexíveis podem ser armazenados. Este é o fundamento da abordagem PNRD. Além disto, a etiqueta também terá a informação de onde ela se encontra dentro do processo, para isto, é armazenado também o vetor de marcação M_k . Dependendo da memória da etiqueta, também é possível a adição de informações adicionais para atender as necessidades de cada processo.

É necessário, também, que existam leitores RFID nos pontos de transição e eles carregam o vetor de disparos (u_k) referente àquela transição. De forma que ao identificar uma etiqueta, automaticamente este leitor faz o cálculo do vetor M_{k+1} e o salva na etiqueta.

Uma característica importante da abordagem PNRD são as exceções. A PNRD é um ótimo recurso para o controle de processos, pois quando há a ocorrência de um erro, o mesmo é identificado imediatamente pelo leitor RFID, pois haverá o surgimento de um valor negativo no vetor de marcações. Para exemplificar, utilizando o exemplo da Figura 4 aplicado a Equação 2.1, partindo do estado M_0 e aplicando o vetor de disparo u_1 que se refere a transição T_1 , tem-se:

$$M_{k+1} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 & 0 & 1 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$$

Com a aparição de um número negativo no vetor de marcações, sabe-se que algo errado aconteceu, e o objeto de estudo, dentro do processo representado pela rede de Petri, seguiu por um caminho errado.

Os leitores RFID são responsáveis por identificar a ocorrência de exceções. Em caso de surgimento de um valor negativo dentro do vetor M_{k+1} , uma notificação deve ser enviada ao sistema supervisor para que o erro seja tratado.

A utilização da PNRD para controle processos se torna viável tendo em vista que a ocorrência de falhas no decorrer destes processos é apontada imediatamente pelos leitores, isto reduz a necessidade de troca de mensagens entre os leitores e os processadores centrais, reduzindo os tempos para troca de mensagens e portanto reduzindo o tempo envolvido neste procedimento.

Na Figura 5 está representado um exemplo de utilização de rede de Petri em um processo de verificação de mancais. Ao fixar uma etiqueta RFID na peça, e utilizando a abordagem PNRD é possível fazer o controle deste processo. Nesta situação, cada lugar corresponde a uma etapa no processo de verificação, as transições são disparadas por leitores RFID e em caso de exceção, ela é apontada imediatamente pelo microcontrolador associado àquele leitor.

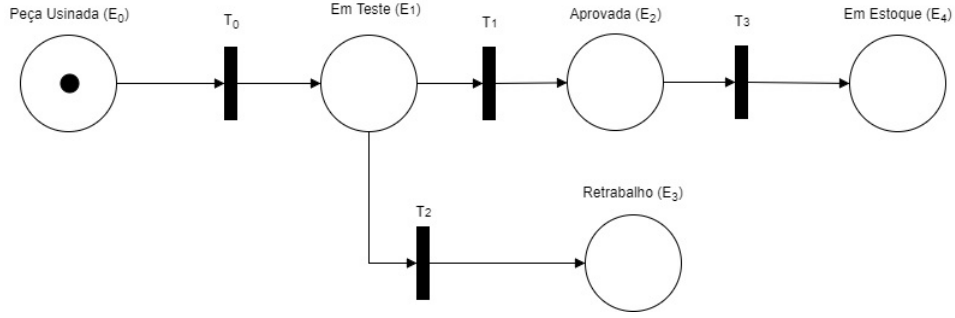


Figura 5 – Representação simplificada de um processo de verificação de mancais usando redes de Petri

Fonte: Silva et al. (2017)

Utilizando os princípios abordados na seção 2.3 pode-se montar a matriz de incidência desta rede de Petri, a qual será armazenada na etiqueta, juntamente com o seu estado atual dentro da rede.

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

Neste exemplo existem três máquinas. A primeira máquina é responsável por cadastrar a peça no teste, ou seja, se existe uma peça usinada, ela dispara uma transição (T_0) e a coloca no estado "Em teste". Depois disto, existe a segunda máquina, responsável por fazer a avaliação da peça, nesta máquina um operário avalia a peça, e pode a aprovar ou reprovar, onde cada uma destas situações tem uma transição correspondente. Após a aprovação a peça parte para a terceira máquina, que gerencia o estoque. Esta ultima etapa garante que nenhuma peça reprovada vá para o estoque.

2.5 PNRD INVERTIDA (IPNRD)

Segundo Fonseca et al. (2018) a PNRD tem como objetivo o monitoramento de agentes passivos, como itens comerciais, peças e produtos. Informações acerca do modelo comportamental (matriz de incidência A^T e vetor de marcação M_k) são armazenados em uma etiqueta passiva fixada ao objeto em questão, enquanto as informações referentes ao vetor de disparos u_k estão associadas aos leitores e antenas distribuídos no ambiente.

Em situações de busca e salvamento, os agentes que requerem identificação são ativos, como usuários, vítimas e socorristas. Neste caso percebe-se que a estrutura clássica

da PNRD é incapaz de controlar o comportamento dos mesmos, isso porque agentes ativos possuem comportamento autônomo e seus modelos comportamentais devem ir além do controle de movimentação ao longo dos pontos estratégicos.

Como os agentes ativos são normalmente microcontrolados, torna-se inviável associar sua identificação a uma etiqueta, sendo mais razoável a utilização destas para a identificação de pontos de interesse no espaço, além do mais, em ambientes remotos existe a dificuldade de manter fontes de energias para os sensores, enquanto os sensores podem ser facilmente acoplados aos agentes ativos.

Esta é a base para a definição da *i*PNRD, nela, há uma inversão dos locais de armazenamento dos termos da Equação 2.1. Nesta abordagem, a matriz de incidência A^T e o vetor de marcações M_k são armazenados no leitor, e na etiqueta são armazenadas informações do vetor de disparo u_k , conforme apresentado na Figura 6.

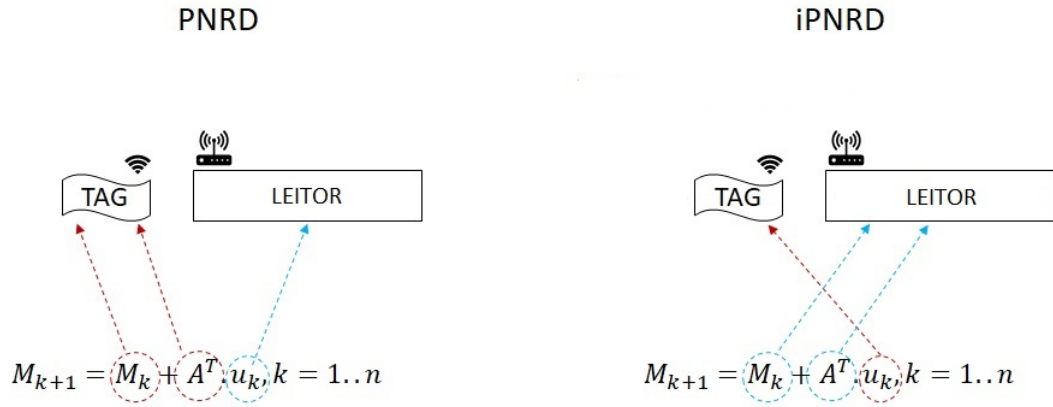


Figura 6 – Associação dos termos da Equação 2.1 com os elementos onde serão armazenados nas abordagens PNRD e *i*PNRD.

Fonte: Fonseca et al. (2018)

2.6 ESTUDO DE CASO: MUNDO DE BLOCOS

O mundo de blocos consiste em um número finito de blocos empilhados em torres sobre uma mesa, o objetivo é mover os blocos de uma posição atual, para uma desejada. Neste problema algumas regras devem ser observadas.

- Somente um bloco pode estar diretamente sobre o outro
- Só um bloco pode ser movido por vez

Neste trabalho será utilizado um exemplo com três blocos, que podem ser distribuídos em três colunas. Ao se discretizar o espaço disponível em posições, pode-se montar uma matriz representando o espaço de três linhas e três colunas, conforme a Figura 7.

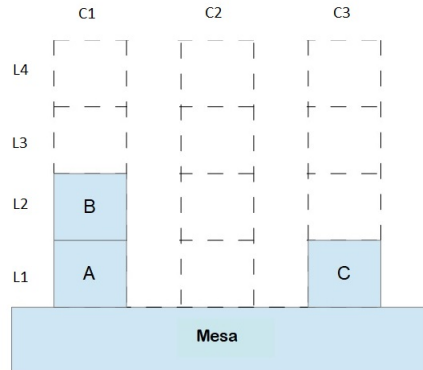


Figura 7 – Matriz representando o espaço de trabalho discretizado.

Fonte: (TAVARES; SOUZA, 2019)

Percebe-se que na Figura 7 existem quatro linhas, isso se faz necessário neste projeto pois movimentações na matriz 3x3 estariam sujeitas a ocorrência de colisões, então, ao adicionar uma nova linha, e delimitando as movimentações horizontais a ocorrerem nela, problemas com colisões são evitados.

No mundo de blocos clássico, não importa a posição das colunas sobre a mesa. Nas condições que serão trabalhadas, se esta característica fosse mantida, seriam possíveis 60 posições distintas dentro desta matriz com os três blocos. Na intenção de reduzir isto, será delimitado que o bloco A só pode estar sobre a mesa na coluna 1, o bloco B só pode estar sobre a mesa na coluna 2 e o bloco C só pode estar sobre a mesa na coluna 3. Isto reduz o total de posições possíveis para 13, o que simplifica o desenvolvimento do trabalho.

No decorrer do texto, é usada a seguinte notação para se referir ao posicionamento dos blocos:

- AB significa que o bloco A se encontra diretamente sobre o bloco B.
- A_B significa que os blocos A e B se encontram em colunas distintas.

Portanto, se temos BA_C, significa que o bloco B se encontra sobre o bloco A em uma coluna, e o bloco C isolado em outra. Este exemplo está representado na Figura 7.

3 METODOLOGIA E DESENVOLVIMENTO

Este trabalho aborda a utilização de PNRD integrada com *i*PNRD para o planejamento automático de soluções para um problema no mundo de blocos proposta por Tavares e Souza (2019). O problema em questão, consiste em três blocos empilhados, cada um com uma identificação única (etiquetas RFID acopladas). O sistema desenvolvido deve ser capaz de reconhecer como estão arranjados, e então planejar, a partir de um servidor WEB, uma rotina de movimentações para que eles fiquem na posição desejada, após isto, deve executar esta rotina e manter as informações das etiquetas atualizadas acerca da sua situação dentro do processo de reorganização. O agente ativo será um braço robótico, que tem como juntas, servo-motores, que serão controlados via Arduino Mega. Ao longo deste capítulo são apresentadas todas as soluções implementadas neste projeto, e na Figura 8 está uma representação do sistema.

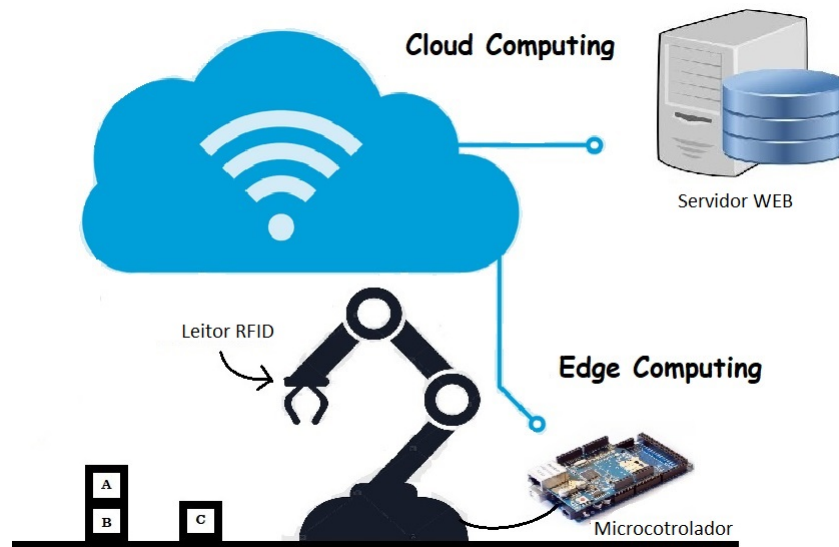


Figura 8 – Sistema Ciber-Físico para solução do Mundo de Blocos utilizando a integração de PNRD e *i*PNRD

3.1 ESPECIFICAÇÃO DA SOLUÇÃO

Parte do sistema funcionará em um servidor WEB, a outra parte, em um microcontrolador local que irá comandar o braço robótico. Todo o planejamento, ou seja, a definição da movimentação do robô e também a atualização das etiquetas RFID será comandada pelo servidor WEB.

O sistema local será responsável por fazer a requisição do planejamento ao servidor

WEB, receber as informações de comando e executá-las, fazendo, inclusive, o tratamento de erros, caso ocorram.

3.1.1 Sistema em nuvem

O sistema em nuvem é responsável por fazer o planejamento das ações do braço robótico, ou seja, ele precisa ser capaz de determinar quais blocos precisam ser movidos, e além disto, precisa determinar como será feita esta movimentação. Para definir quais blocos precisam ser movidos, ou seja, a solução prática do problema do Mundo de Blocos, foi projetada uma rede de Petri onde cada estado se refere a um arranjo, e as transições são rearranjos movendo-se somente um bloco, conforme apresentado na Figura 9. Após isto, o sistema deve ser capaz de identificar como o braço robótico irá executar esses movimentos, para isto foi utilizada outra rede de Petri, apresentada na Figura 10. Nela, cada lugar da rede de Petri está relacionado com uma posição no espaço de trabalho.

A Figura 9 apresenta o modelo do problema do Mundo de Blocos em rede de Petri, percebe-se que neste modelo estão contemplados todos os blocos, ou seja, esta é a perspectiva do agente ativo. Neste modelo, cada estado se trata de um arranjo dentro do Mundo de Blocos, e as transições são mudanças de lugar de um único bloco, estes elementos estão descritos no Apêndice C.

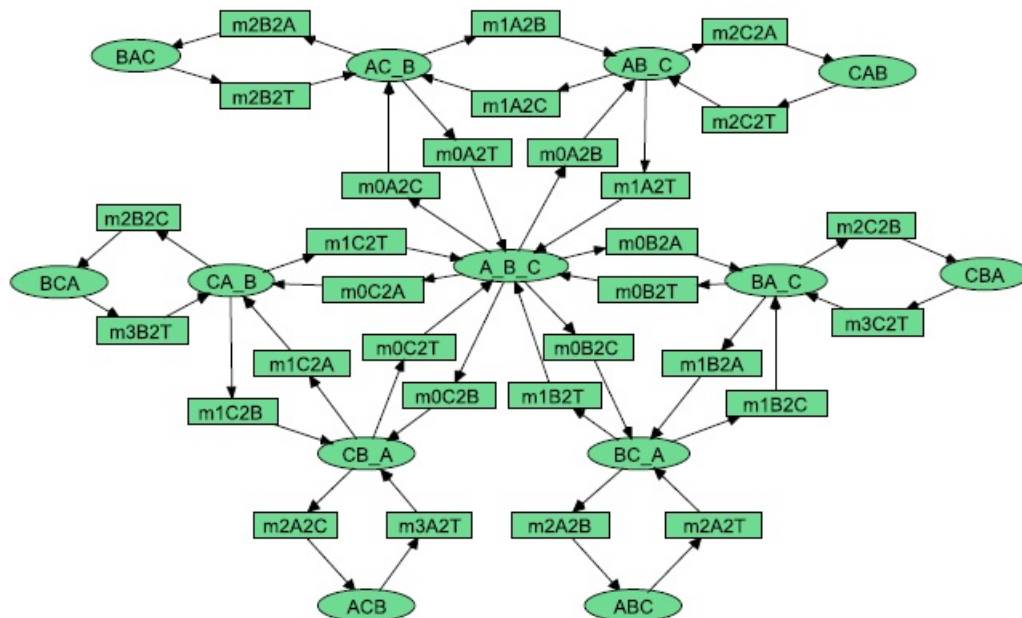


Figura 9 – Modelo *iPNRD*

Fonte: (TAVARES; SOUZA, 2019)

É desejável que o sistema encontre a solução que tenha o menor custo de tempo e de energia, logo, se estamos em um estado M_x e queremos ir para um estado M_y deve-se procurar o caminho que resulte na ativação do menor número de transições possível.

Como citado na seção 2.3, as redes de Petri são grafos orientados, ou seja, mecanismos de busca em grafos são ideais para se encontrar a solução ótima para o problema.

Neste trabalho, será utilizada uma busca em largura, na qual são analisados todos os estados adjacentes ao inicial, e posteriormente os que estão adjacentes eles, seguindo uma fila onde os primeiros adicionados serão os primeiros a serem analisados. Esta análise consiste, simplesmente, em compará-los com o estado desejado.

Após encontrado o estado desejado, faz-se o caminho contrário, e então, são determinadas todas as transições que precisam ser ativadas para alcançá-lo.

São conhecidas, neste ponto, as transições da rede *iPNRD*, ou seja, quais blocos devem ser movimentados e para onde. Porém, é necessário ainda, definir como serão executadas estas transições, ou seja, os movimentos do braço robótico no espaço físico. Para isto, foi modelada outra rede de Petri, representada na Figura 10, esta rede irá representar o espaço físico de trabalho. Neste sentido, as posições no espaço serão os estados desta rede de Petri. No decorrer do trabalho, esta rede será citada como Espaço de Petri.

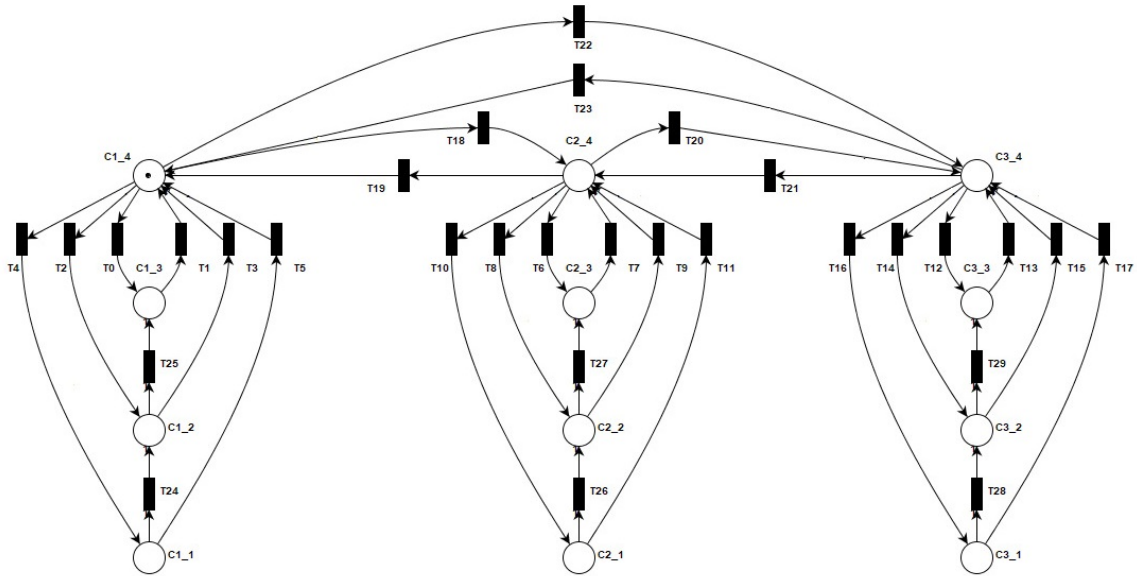


Figura 10 – Espaço de Petri

Fonte: (TAVARES; SOUZA, 2019)

Ao se associar cada transição *iPNRD* a uma posição inicial e uma posição final dentro do Espaço de Petri, pode-se conduzir novamente uma busca em largura para definir

o caminho que o braço robótico irá percorrer ao movimentar os blocos. Nota-se que as transições da *iPNRD* se tratam das movimentações do robô com a garra fechada, ou seja, segurando o bloco. É necessário também definir a movimentação do robô com a garra aberta, ou seja, uma movimentação que vai ocorrer tendo como estado inicial, no Espaço de Petri, a posição final de uma transição da *iPNRD*, e tendo como estado final, do Espaço de Petri, a posição do início da próxima transição *iPNRD*. Lembrando também de fazer uma movimentação inicial que vai da posição padrão de início, para a posição inicial da primeira transição *iPNRD*, e também, uma movimentação final, que vai da posição final da ultima transição *iPNRD* até a posição padrão de início.

Também é implementado um controle de erros que será executado pelos leitores RFID a partir da abordagem PNRD. No decorrer das movimentações, os estados das etiquetas acerca da sua posição na rede de Petri apresentada na Figura 11 serão atualizados, ou seja, ao se concluir cada transição *iPNRD*, existe uma transição PNRD relacionada que será disparada. É importante que a informação de qual transição PNRD deve ser disparada seja enviada ao sistema local, este, estará efetuando os disparos e em caso de exceção irá tratar do erro. Os elementos da Figura 11 estão descritos no Apêndice D

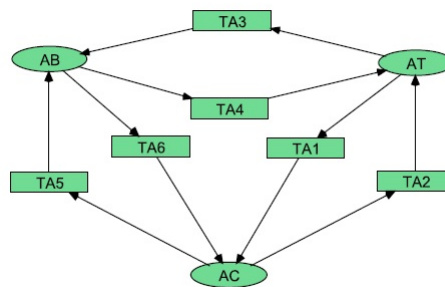


Figura 11 – Modelo PNRD (perspectiva do bloco)

Fonte: (TAVARES; SOUZA, 2019)

Após definidos os momentos de se fechar e abrir a garra, que devem ocorrer no início e fim de cada transição *iPNRD*, respectivamente, está completo o planejamento de ações, então deve-se criar uma mensagem com esta informação para ser enviada ao sistema local. Um fluxograma apresentando as principais tarefas do servidor WEB estão representadas na Figura 12.

O servidor foi hospedado em um computador com Intel(R) Core(TM) i5-3470 CPU @ 3.20GHz x64, 16,0 GB RAM. Foi utilizado um servidor do tipo Apache, programado em linguagem PHP. O programa utilizado consta no Apêndice A.

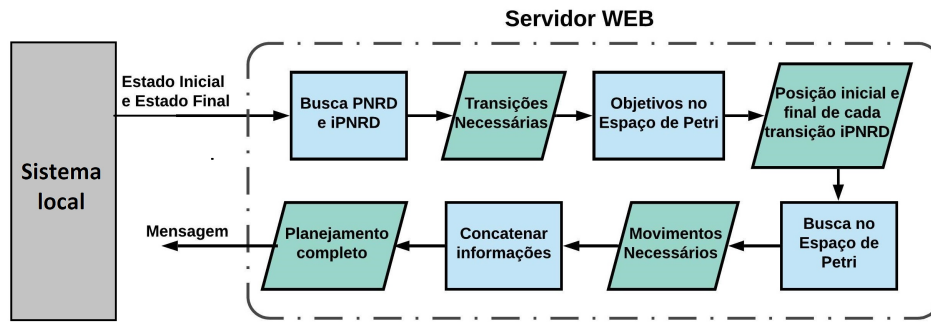


Figura 12 – Planejamento de processos a eventos discretos utilizando rede de Petri em Cloud Computing

3.1.1.1 Exemplo

Para exemplificar o funcionamento, consideremos um uma situação onde deseja-se ir do estado A_B_C para o estado AB_C, ou seja, será movido o bloco A que está sobre a mesa para que o mesmo fique sobre o bloco B. Ao receber estes estados, o sistema em nuvem irá efetuar a busca em largura e determinar que a transição "m0A2B" representada na Figura 9 deve ser ativada para se atingir o estado desejado.

Cada transição da *iPNRD* corresponde a um par de posições no Espaço de Petri. Neste caso específico, a posição inicial será a posição que o bloco A se encontra, e a posição final será em cima do bloco B, ou seja, trazendo isto para o Espaço de Petri representado na Figura 10, sabe-se que a posição inicial é a C1_1 e a posição final é a C2_2. Tendo em mãos estas posições, o programa executa novamente uma busca em largura no Espaço de Petri, definindo assim as transições necessárias para se mover de um estado para o outro.

Então, a busca em largura determina que as transições para se atingir a posição desejada são: $T24 \rightarrow T25 \rightarrow T1 \rightarrow T18 \rightarrow T8$. Após definidas as transições, basta traduzi-las em movimentações pelo sistema local.

Então, primeiro o robô se encaminhar da posição padrão até a posição C1_1 que é a inicial e após sair da posição final, C2_2, ele deve voltar para a posição padrão que é a C2_4.

3.1.2 Sistema local

No sistema local, desde a leitura das etiquetas até a efetuação dos disparos, todas as funções que envolvam interação com as etiquetas RFID foram feitas utilizando as bibliotecas desenvolvidas por Silva et al. (2017) em seu trabalho. Entretanto, foi feita uma modificação, onde foi inserida uma informação adicional nas etiquetas, desta forma, além de a etiqueta ter informações de em qual estado PNRD ela se encontra, ela também contém

o estado desejado. Desta forma, os blocos controlam todo o processo sem interferência de informações vindas do agente ativo.

Ao ser iniciado, o braço robótico deve primeiramente fazer uma varredura do Espaço de Petri. Como observado na figura Figura 13, o braço possui, em cima da garra (2), um modulo leitor RFID (1) do tipo NFC PN532, que será responsável por adquirir as informações de identificação dos blocos (3) a partir das etiquetas. Tendo as informações de qual bloco se encontra em qual lugar do espaço de Petri, é possível definir o estado inicial e o estado desejado dos blocos.

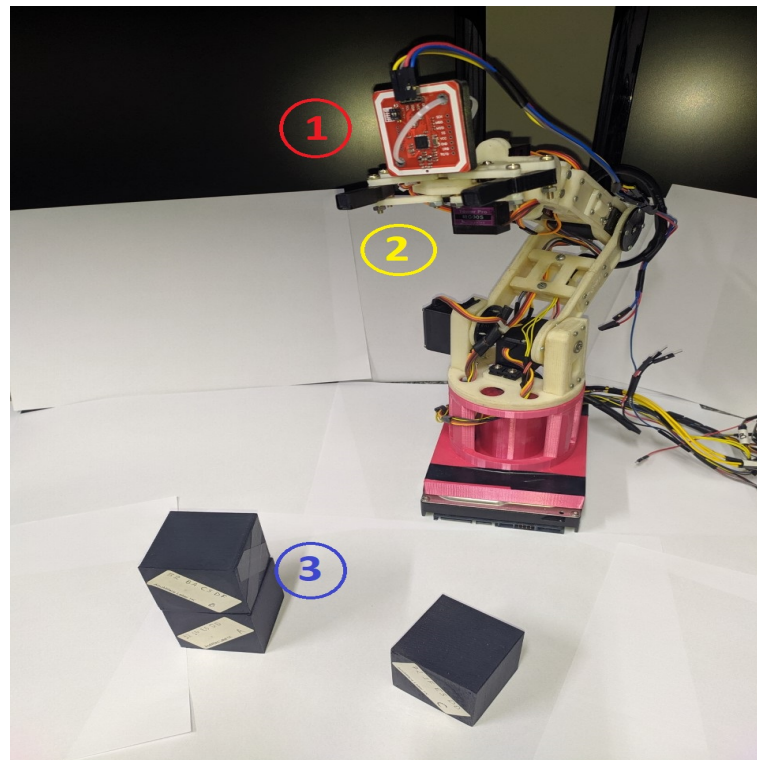


Figura 13 – Braço robótico equipado com leitor RFID ao lado dos blocos etiquetados

Determinados estes estados, o sistema faz uma requisição para o servidor, na qual ele informa o estado atual e o desejado. A comunicação entre o sistema local e o servidor é feita via ethernet, utilizando em conjunto com o Arduino Mega um módulo Ethernet W5100, apresentado na Figura 14. O sistema em nuvem faz o processamento descrito na subseção 3.1.1 e responde com o planejamento de ações do robô.

A requisição é efetuada seguindo o protocolo HTTP, que se da no seguinte padrão:

```
GET http://192.168.0.124/index.php?i=8&f=10 HTTP/1.1  
Host: 192.168.0.124
```

Sendo a primeira linha referente ao método "GET" utilizado e ao endereço URL, a segunda ao endereço IP do servidor. Os componentes do endereço URL estão explicados

na Tabela 1.

Tabela 1 – Componentes da requisição

Endereço do servidor <code>http://192.168.0.124/</code>	Arquivo do software <code>index.php?</code>	Estados inicial e final <code>i=8&f=10</code>
--	--	--

Como já citado, o braço robótico é movido via servo-motores. Uma maneira de simplificar a atuação do mesmo, foi pré-configurar posições de angulação dos servos referentes a cada estado no espaço de Petri, assim, ao receber os estados do espaço de Petri pelos quais o robô deve passar, basta mandar seus servos assumirem aquela posição pré-definida.

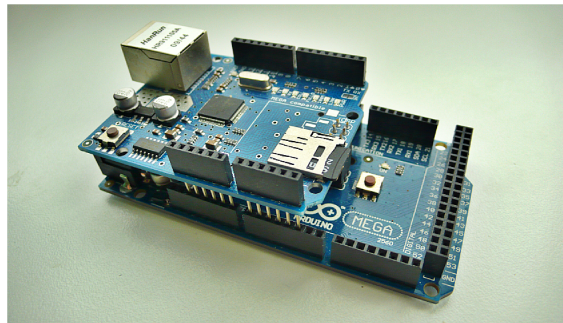


Figura 14 – Arduino Mega acoplado ao módulo Ethernet W5100

Então, ao receber a mensagem do servidor, com as informações de por quais estados do espaço de Petri o braço deve passar, o microcontrolador comanda a movimentação do robô, fechando e abrindo a garra quando necessário.

No meio da mensagem de movimentação do braço, existem, também, informações acerca qual transição PNRD deve ser disparada nos momentos de abertura da garra, ou seja, de encerramento de uma transição i PNRD. O sistema deve ser capaz de fazer o disparo, e também analisar se ele ocorreu de forma correta ou se houve exceção.

Em caso de exceção detectada, sabe-se que ocorreu algum erro. Então o sistema deve parar o que está fazendo, e reiniciar seus procedimentos, partindo da varredura inicial, e refazendo a conexão com o servidor para receber um novo planejamento de ações. Um fluxograma com os passos de funcionamento do sistema local está representado na Figura 15 e o código na linguagem C++ consta no Apêndice B.

3.2 TESTES

Para fins de avaliação de desempenho, o sistema foi submetido a diversas situações diferentes. A seguir está descrita a realização de um destes testes.

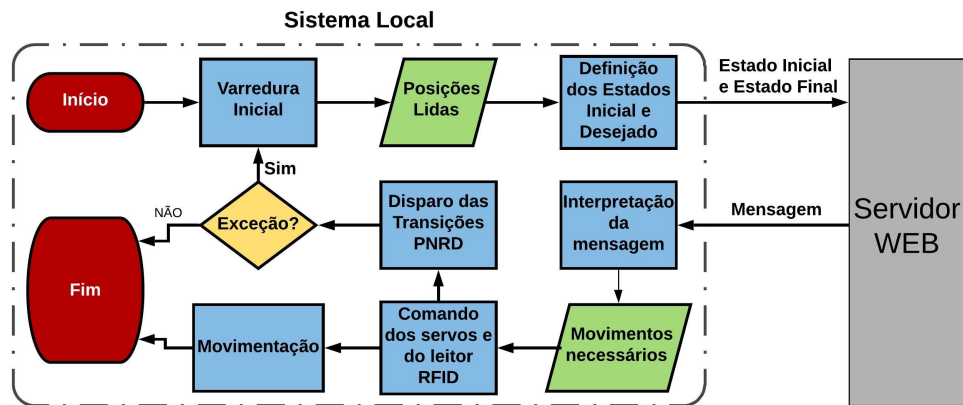


Figura 15 – Fluxograma de ações do sistema local

Entre dois estados podem existir diversas transições, para se testar a velocidade de processamento do sistema, foi escolhido um problema onde se tinha o máximo número de transições entre os estados, ou seja, 4 transições. Os estados em questão são ABC e BAC. Na Figura 16 está representado o planejamento ideal para a solução do problema. Se o sistema oferecer este mesmo planejamento, então sabe-se que ele está operando de forma eficiente.

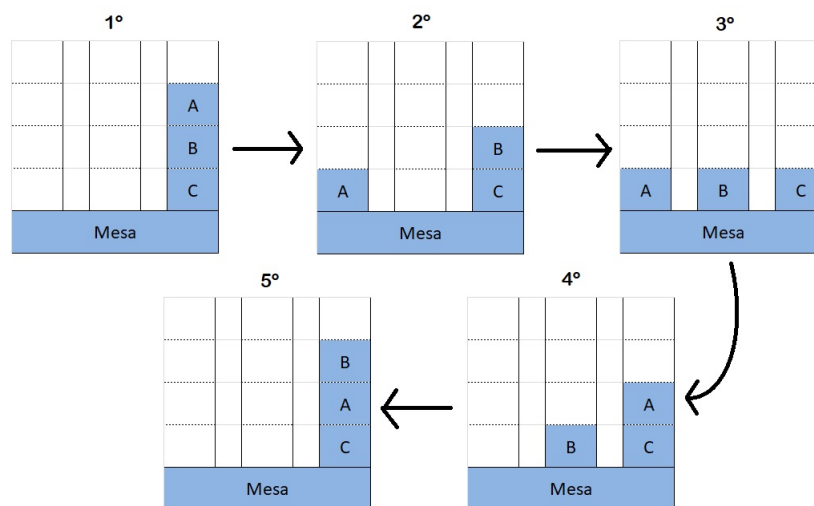


Figura 16 – Planejamento ideal

Para fins de comparação, foi implementado um sistema de planejamento local, no Arduino, e foram colhidos tempos de processamento dos dois sistemas, tanto em nuvem quanto local.

O Arduino Mega 2560 possui uma memória RAM de 8 Kb, enquanto que a memória

dedicada para o servidor WEB foi configurada em 128 MB. É de se esperar que, o servidor WEB tenha uma resposta mais rápida, já que nele não será necessária a ocorrência dos *memory swaps*, que ocorrem quando o sistema precisa esvaziar a memória que não está sendo utilizada para carregar mais dados. Porém, existem os atrasos devido a conexão e troca de dados entre os sistemas local e WEB.

Os tempos serão obtidos a partir do comando `millis()` da Arduino IDE, o qual tem resolução de 1 *ms*. A partir deste comando, é possível calcular tempos de processamento de certas partes do código. Serão contabilizados os tempos em cada caso conforme a Tabela 2, e serão feitas 30 medições.

Tabela 2 – Tempos para medição

Sistema Local	T_p
Sistema em nuvem	$T_c + T_m + T_p$

Onde:

- T_p = Tempo de planejamento
- T_c = Tempo de estabelecimento de comunicação
- T_m = Tempo de troca de mensagens

Além destes testes, para avaliar como o robô opera em caso de exceção, foram efetuadas movimentações dos blocos manualmente enquanto ele operava, assim seria possível ver na prática, o que ocorre quando ele tenta disparar uma transição no bloco errado.

4 RESULTADOS E DISCUSSÃO

Primeiro foi analisado se o sistema faria o planejamento correto. A Figura 17 apresenta o planejamento do sistema, a representação segue a seguinte notação: $E\{X\}: Y$, onde E significa Estado, X corresponde a etapa no processo, e Y é relativo a numeração do estado atual, onde ABC é o estado 8, e BAC é o estado 10. Percebe-se que o planejamento foi idêntico ao ideal, apresentado na Figura 16. Portanto, a nível lógico, o sistema funciona de forma apropriada.

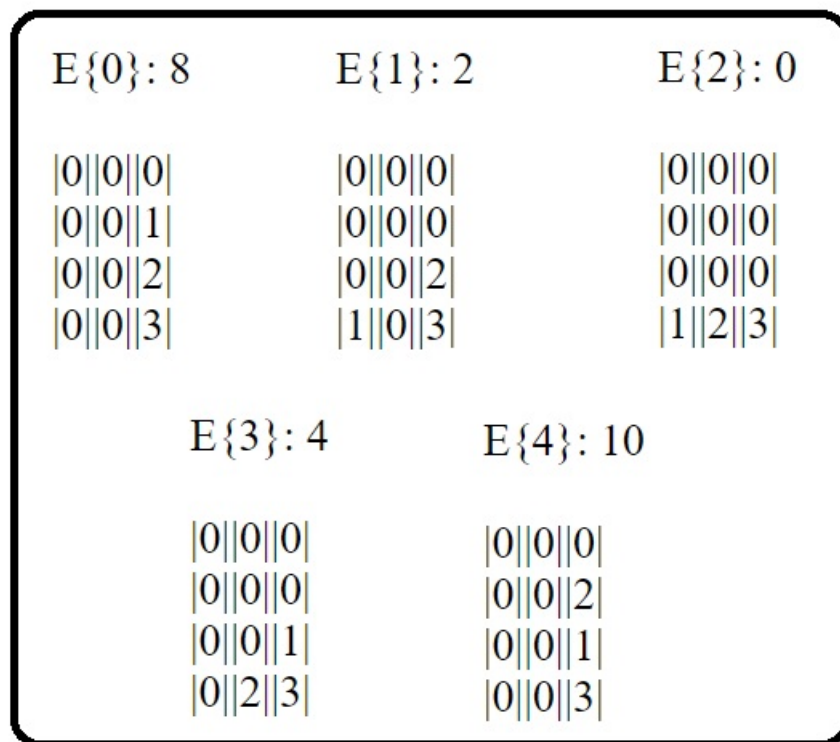


Figura 17 – Planejamento real

Na Figura 18 está apresentada a mensagem de resposta do servidor WEB, nela existem letras e números, onde cada um tem um significado a ser interpretado pelo sistema local. Cada letra corresponde a uma posição no Espaço de Petri, e cada número a uma transição PNRD que precisa ser disparada naquele exato momento. Os detalhes desta mensagem estão apresentados no Apêndice E. As requisições HTTP recebem como resposta uma grande quantidade de informações, que não são interessantes nesta aplicação. Para facilitar a localização da informação útil dentro da resposta do servidor, esta mensagem está contida entre um asterisco e uma vírgula.

Com relação aos tempos de processamento, os resultados obtidos estão apresentados

*WOQXRPHFDBW1ACEGOQSXTRPIKMW1NLJHFDBXACEGOQSW2TRPIKMXNLJOQW0RP,

Figura 18 – Mensagem com o planejamento de ações

na Tabela 3. Não foi informado um intervalo de confiança pois todos os resultados foram idênticos. Isso demonstra que este intervalo se encontra abaixo da resolução do sistema de medição.

Tabela 3 – Tempos de resposta

Tempo de resposta do sistema WEB	33 <i>ms</i>
Tempo de resposta do sistema local	107 <i>ms</i>

Percebe-se que o sistema local exige um tempo de processamento maior. As medidas obtidas neste caso correspondem a 324,24% do valor obtido no sistema WEB. Porém, ambos os resultados foram satisfatórios, tendo em vista que, para este caso específico, estes tempos de planejamento são insignificantes, já que o braço utiliza vários segundos para executar as movimentações.

Não se pode ignorar o fato que, se extrapolarmos para largas escalas, em uma situação onde se requer maior capacidade computacional e não há um processo seguinte ao planejamento que funcione como um gargalo, estes resultados se tornariam relevantes, já que estes planejamentos, dependendo da aplicação, se realizariam várias vezes durante um dia, e no final, o tempo total consumido seria significativo.

O maior problema da abordagem que utiliza Internet das Coisas foi a conexão. Por diversas vezes o sistema local não conseguia fazer a conexão com o sistema em nuvem. Mesmo implementando isto numa rede local, onde só tinha o servidor, alojado em um computador e o microcontrolador fazendo a requisição. É possível que este problema se dê pela baixa qualidade do módulo utilizado. Porém, não se pode ignorar os problemas provenientes das comunicações, que podem ocorrer mesmo em um sistema mais robusto.

Uma maneira de lidar com este problema, foi fazer a integração dos dois sistemas. Fazendo um sistema de contingenciamento, onde em caso de falha na conexão, o sistema local realiza o planejamento da movimentação do braço robótico.

A respeito do sistema local, este não apresentou falhas significativas do ponto de vista de programação e execução das ordens. Porém, após ter completado o projeto percebeu-se que o modelo de braço robótico utilizado não foi o ideal, pois sua garra não possui força o suficiente para segurar o bloco sem danificar o servo-motor encarregado desta junta. Além disto, ao fechar a garra, ela acaba empurrando o bloco alguns milímetros para frente, o que compromete a execução das ordens, pois ao tentar pegar um bloco que

se encontra em cima de outro, ele empurra o de baixo também, e ao tentar pegar o que estava embaixo ele já não se encontra na posição correta. A melhor maneira de se resolver isto seria utilizar um braço robótico com outro mecanismo na garra, uma que se feche sem empurrar o bloco.

Também foi possível observar o comportamento do sistema ao se realizar a troca deliberada de blocos durante a movimentação. Ao tentar disparar a transição no bloco errado o leitor RFID identifica a ocorrência de exceção e imediatamente o braço robótico cessa a rotina e recomeça a partir da varredura, detectando a nova posição dos blocos e fazendo novamente a solicitação ao servidor WEB.

5 CONCLUSÃO

A exceção das limitações físicas oferecidas pelo braço robótico, pode-se considerar que este trabalho atingiu seus objetivos. Foi possível a implementação de dois sistemas distintos, um operando em nuvem e outro localmente, trabalhando simultaneamente para se atingir um objetivo.

Foi possível também se observar os conceitos da Indústria 4.0 sendo aplicados em um projeto onde se utilizou da Internet das Coisas, da Computação em Nuvem, das tecnologias RFID, integração de sistemas, etc. E isto proporcionou a montagem de um sistema capaz de trabalhar autonomamente onde os próprios objetos (blocos etiquetados) podiam determinar o que seria feito com eles, enquanto o sistema projetado pode ler esta informação, fazer o planejamento, e operar sozinho, sem interferência humana.

A abordagem PNRD/*i*PNRD se mostrou eficaz quando utilizada para a resolução de sistemas complexos. Tanto utilizando um sistema em nuvem quanto um servidor local foi obtida uma rápida resposta. Além disto, quando ocorrem, as falhas são identificadas instantaneamente em nível local, sem necessidade de processamento, e são tratadas automaticamente, conferindo autonomia ao sistema.

Também pode-se concluir que em casos onde há a necessidade de um grande poder computacional, é mais rápido operar com um sistema em nuvem, desde que se possua um equipamento de comunicação robusto para evitar falhas de comunicação. A integração do sistema de contingenciamento mencionado na sessão anterior se torna uma ótima alternativa para situações em que não se possui equipamentos robustos de comunicação. Ela garante que o sistema funcione continuamente, e em caso de falhas na rede ele só fica mais lento ao invés de parar completamente.

Em problemas mais simples, onde se exige menor capacidade de processamento, é mais viável fazer o tratamento na fronteira, ou seja, utilizando um microcontrolador local, desta forma, evitam-se falhas de comunicação. Mesmo que a resposta não seja obtida tão rápido quanto no sistema em nuvem, ela ainda é obtida em um curto intervalo de tempo, e o sistema se torna bem mais confiável, pois poucas coisas podem levar a falha.

Neste caso utilizou-se somente um cliente e um servidor. Uma forma de melhorar isto, em um trabalho futuro, seria a utilização de vários clientes e um servidor. Para isto, seria interessante a implementação de um sistema baseado no protocolo de comunicação MQTT, desenvolvido para auxiliar nas implementações de projetos envolvendo Internet das Coisas.

REFERÊNCIAS

- AALST, W. M. Van der. The application of petri nets to workflow management. **Journal of circuits, systems, and computers**, World Scientific, v. 8, n. 01, p. 21–66, 1998. Citado na página 19.
- AHSAN, K.; SHAH, H.; KINGSTON, P. Rfid applications: An introductory and exploratory study. **arXiv preprint arXiv:1002.1179**, 2010. Citado na página 15.
- ASHTON, K. et al. That ‘internet of things’ thing. **RFID journal**, v. 22, n. 7, p. 97–114, 2009. Citado na página 15.
- BROCK, D. L. The electronic product code (epc). **Auto-ID Center White Paper MIT-AUTOID-WH-002**, p. 1–21, 2001. Citado na página 14.
- FINKENZELLER, K. **RFID handbook: fundamentals and applications in contactless smart cards, radio frequency identification and near-field communication**. [S.l.]: John Wiley & Sons, 2010. Citado nas páginas 14 e 15.
- FONSECA, J. P. d. S. et al. Redes de petri de alto nível e pnrđ invertida associadas ao controle de robôs móveis: uma abordagem para operações de busca e salvamento em trilhas e travessias. Universidade Federal de Uberlândia, 2018. Citado nas páginas 21 e 22.
- GILCHRIST, A. **Industry 4.0: the industrial internet of things**. [S.l.]: Apress, 2016. Citado na página 16.
- LASI, H. et al. Industry 4.0. **Business & information systems engineering**, Springer, v. 6, n. 4, p. 239–242, 2014. Citado na página 11.
- LIU, X. et al. An energy-aware offloading framework for edge-augmented mobile rfid systems. **IEEE Internet of Things Journal**, IEEE, 2018. Citado na página 12.
- MACIEL, P. R.; LINS, R. D.; CUNHA, P. R. **Introdução às redes de Petri e aplicações**. [S.l.]: UNICAMP-Instituto de Computacao, 1996. Citado nas páginas 16 e 17.
- MELL, P.; GRANCE, T. et al. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National . . . , 2011. Citado na página 11.
- MURATA, T. Petri nets: Properties, analysis and applications. **Proceedings of the IEEE**, IEEE, v. 77, n. 4, p. 541–580, 1989. Citado na página 16.
- SILVA, C. E. A. d. et al. Desenvolvimento de biblioteca para aplicações de pnrđ e pnrđ invertida embarcadas em arduino. Universidade Federal de Uberlândia, 2017. Citado nas páginas 18, 21 e 28.
- SUNDMAEKER, H. et al. Vision and challenges for realising the internet of things. **Cluster of European Research Projects on the Internet of Things, European Commision**, v. 3, n. 3, p. 34–36, 2010. Citado na página 11.
- TAURION, C. **Cloud computing-computação em nuvem**. [S.l.]: Brasport, 2009. Citado na página 11.

TAVARES, J. J.-P. Z. D. S.; SARAIVA, T. A. Elementary petri net inside rfid distributed database (pnrd). **International Journal of Production Research**, Taylor & Francis, v. 48, n. 9, p. 2563–2582, 2010. Citado na página 19.

TAVARES, J. J.-P. Z. d. S.; SOUZA, G. d. A. Pnrd and ipnrd integration assisting adaptive control in block world domain. **Proceedings of the International Workshop on Petri Nets and Software Engineering 2019**, p. 73–90, 2019. Citado nas páginas 23, 24, 25, 26 e 27.

APÊNDICE A – PROGRAMA DO SERVIDOR WEB

Programa na linguagem PHP hospedado no servidor.

```
<?php
//Recebe os estados inicial e final a partir da requisicao HTTP
$EstadoInicial = $_GET['i'];
$EstadoFinal = $_GET['f'];

$ESTADOS = [
    [// A|B|C
        [0,0,0],
        [0,0,0],
        [0,0,0],
        [1,2,3]
    ],
    [// AB|C
        [0,0,0],
        [0,0,0],
        [2,0,0],
        [1,0,3]
    ],
    [// A|CB
        [0,0,0],
        [0,0,0],
        [0,0,2],
        [1,0,3]
    ],
    [// BA|C
        [0,0,0],
        [0,0,0],
        [0,1,0],
        [0,2,3]
    ],
    [// B|CA
        [0,0,0],
        [0,0,0],
        [0,0,1],
        [0,2,3]
    ],
    [// AC|B
        [0,0,0],
        [0,0,0],
        [3,0,0],
        [1,2,0]
    ],
    [// A|BC
```


[illegible]

```

];
$MovimentoTrack = [
[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
];
$FILAPosicao = [
[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
];

$INTERMEDIO = [[-1,-1],[-1,-1],[-1,-1],[-1,-1],[-1,-1],[-1,-1],[-1,-1],[-1,-1],
[-1,-1],[-1,-1]];
$REPETE = 0;
$GARRA = [0,1,0,1,0,1,0,1,0];

//Parte do programa que executa a busca em largura
while( $FIMEstado == 0){ //enquanto nao encontrou o estado final:
for ( $i=0; $i<30; $i++){
    if ($TRANSCOE[$EstadoAtual][$i] == -1){
        for($j=0; $j<13; $j++){
            if($TRANSCOE[$j][$i] == 1){
                if($EstadoTrack[$j] == -1)
                {
                    $EstadoTrack[$j] = $EstadoAtual;
                    $TransicaoTrack[$j] = $i;
                }
                if($EstadoAtual == $EstadoFinal)
                {
                    $FIMEstado = 1;
                }
            }
        }
    }
    else
    {
        for($k=0;$k<13;$k++)
        {

```

```

        if($j == $FILAEstado[$k])
        {
            break;
        }
        if($FILAEstado[$k] == -1)
        {
            $FILAEstado[$k] = $j;
            break;
        }
    }
}
}
}
if($FIMEstado == 1){
    break;
}
}

if($FIMEstado == 0){
    $EstadoAtual = $FILAEstado[0];
}
for($i = 0; $i<12; $i++){
    $FILAEstado[$i] = $FILAEstado[$i+1];
}
$FILAEstado[12] = -1;
}

//Neste ponto ja sabe-se as transicoes que precisam ser ativas, em seguida
//elas serao salvas para serem utilizadas futuramente
$p = 5;
$contador = 5;
$q = $EstadoFinal;
while ($q != $EstadoInicial){
    $PAREstado[$p][0] = $q;
    $PAREstado[$p][1] = $TransicaoTrack[$q];
    $q = $EstadoTrack[$q];
    $p--;
}
$PAREstado[$p][0] = $q;
$PAREstado[$p][1] = $TransicaoTrack[$q];
while($PAREstado[0][0] == -1) {
    for($p = 0; $p < (6-1) ; $p++) {
        $PAREstado[$p][0] = $PAREstado[$p+1][0];
        $PAREstado[$p][1] = $PAREstado[$p+1][1];
    }
    $contador--;
}
while(TRUE) {
    for($p=0;$p<(6-1);$p++) {
        $PAREstado[$p][1] = $PAREstado[$p+1][1];
    }
    if($PAREstado[0][1] != -1){

```

```

        break;
    }
}

$PROFUNDIDADE = $contador;
$u = 0;

//Neste ponto o programa associa as transicoes da iPNRD
//com o par de posicoes relacionado a elas

$INTERMEDIO[$u][0] = 4;
$INTERMEDIO[$u][1] = $TRANSFORMADA[0][$PAREstado[0][1]];
$u++;
for($k = 0; $k<$PROFUNDIDADE; $k++) {
    $INTERMEDIO[$u][0] = $TRANSFORMADA[0][$PAREstado[$k][1]];
    $INTERMEDIO[$u][1] = $TRANSFORMADA[1][$PAREstado[$k][1]];
    $u++;
    if ($k < $PROFUNDIDADE -1) {
        $INTERMEDIO[$u][0] = $TRANSFORMADA[1][$PAREstado[$k][1]];
        $INTERMEDIO[$u][1] = $TRANSFORMADA[0][$PAREstado[$k+1][1]];
        $u++;
    }
}

$INTERMEDIO[$u][0] = $TRANSFORMADA[1][$PAREstado[$k][1]];
$INTERMEDIO[$u][1] = 4;

for($i=0; $i<10; $i++) {
    for($j=0; $j<10;$j++){
        for($k = 0; $k<2; $k++){
            $PARPosicao[$i][$j][$k] = -1;
        }
    }
}

while($REPETE < ($PROFUNDIDADE*2) + 1)
{
    $PosicaoInicial[$REPETE] = $INTERMEDIO[$REPETE][0];
    $PosicaoFinal[$REPETE] = $INTERMEDIO[$REPETE][1];
    $PosicaoAtual[$REPETE] = $PosicaoInicial[$REPETE];
    while($FIMPosicao == 0)
    {
        //Neste ponto e feita a busca em largura no Espaco de Petri para
        //definir as movimentacoes necessarias
        for($i=0;$i<22;$i++)
        {
            if($MOVIMENTO[$PosicaoAtual[$REPETE]][$i] == -1)
            {
                for($j=0;$j<12;$j++)
                {
                    if($MOVIMENTO[$j][$i] == 1)
                    {
                        if($PosicaoTrack[$REPETE][$j] == -1)
                        {
                            $PosicaoTrack[$REPETE][$j] = $PosicaoAtual[$REPETE];
                        }
                    }
                }
            }
        }
    }
}

```

```

        $MovimentoTrack[$REPETE][$j] = $i;
    }
    if($PosicaoAtual[$REPETE] == $PosicaoFinal[$REPETE])
    {
        $FIMPosicao = 1;
    }
    else
    {
        for($k=0;$k<12;$k++)
        {
            if($j == $FILAPosicao[$REPETE][$k])
            {
                break;
            }
            if($FILAPosicao[$REPETE][$k] == -1)
            {
                $FILAPosicao[$REPETE][$k] = $j;
                break;
            }
        }
    }
}

if($FIMPosicao == 1)
{
    break;
}

if($FIMPosicao == 0)
{
    $PosicaoAtual[$REPETE] = $FILAPosicao[$REPETE][0];
}
for($i=0;$i<(12-1);$i++)
{
    $FILAPosicao[$REPETE][$i] = $FILAPosicao[$REPETE][$i+1];
}
$FILAPosicao[$REPETE][11] = -1;
}
$REPETE++;
$FIMPosicao = 0;
}

$REPETE = 0;
while($REPETE < (($PROFUNDIDADE*2) + 1))
{
    $p=8;
    $contador=8;
    $q = $PosicaoFinal[$REPETE];
    while($q != $PosicaoInicial[$REPETE])
    {
        $PARPosicao[$REPETE][$p][0] = $q;
    }
}

```

```

        $PARPosicao[$REPETE][$p][1] = $MovimentoTrack[$REPETE][$q];
        $q = $PosicaoTrack[$REPETE][$q];
        $p--;
    }
    $PARPosicao[$REPETE][$p][0] = $q;
    $PARPosicao[$REPETE][$p][1] = $MovimentoTrack[$REPETE][$q];
    while($PARPosicao[$REPETE][0][0] == -1)
    {
        for($p = 0; $p < (10-2) ; $p++)
        {
            $PARPosicao[$REPETE][$p][0] = $PARPosicao[$REPETE][$p+1][0];
            $PARPosicao[$REPETE][$p][1] = $PARPosicao[$REPETE][$p+1][1];
        }
        $contador--;
        $PARPosicao[$REPETE][$p][1] = $PARPosicao[$REPETE][$p+1][1];
    }
    while(TRUE)
    {
        for($p = 0; $p < 10-1; $p++)
        {
            $PARPosicao[$REPETE][$p][1] = $PARPosicao[$REPETE][$p+1][1];
        }
        if($PARPosicao[$REPETE][0][1] != -1)
        {
            break;
        }
    }
    $PROFPOSICAO[$REPETE] = $contador;
    $REPETE++;
}

//Neste ponto temos todas as transicoes da iPNRD e do espaco de Petri,
//entao e chegado o momento de montar a mensagem que sera exibida na tela
$REPETE = 0;
echo "*";
while($REPETE < ($PROFUNDIDADE*2) + 1)
{
    echo chr(87+($REPETE%2));
    if ($REPETE/2>0 && ($REPETE%2) == 0){
        echo $TRANSICAOPNRD[$PAREstado[(($REPETE-1)/2)[1]];
    }
    for($i=0;$i<$PROFPOSICAO[$REPETE];$i++)
    {
        echo chr(65+$PARPosicao[$REPETE][$i][1]);
    }
    $REPETE++;
}
echo ",";
echo "</br></br>";

```

?>

APÊNDICE B – PROGRAMA DO SISTEMA LOCAL

Programa na linguagem C++ hospedado no Arduino Mega.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

//-----
//Matriz de Adjacencia
int TRANSICOES [13][30] =
{
{-1,1,-1,1,-1,1,-1,1,-1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, // 0 - A|B|C
{ 1,-1,0,0,0,0,0,0,0,0,0,0,0,0,-1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, // 1 - AB|C
{0,0,1,-1,0,0,0,0,0,0,0,0,0,1,-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, // 2 - A|CB
{0,0,0,0,1,-1,0,0,0,0,0,0,0,0,-1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, // 3 - BA|C
{0,0,0,0,0,0,1,-1,0,0,0,0,0,0,1,-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, // 4 - B|CA
{0,0,0,0,0,0,0,0,1,-1,0,0,0,0,0,0,-1,1,0,0,0,0,0,0,0,0,0,0,0,0}, // 5 - AC|B
{0,0,0,0,0,0,0,0,0,0,1,-1,0,0,0,0,1,-1,0,0,0,0,0,0,0,0,0,0,0,0}, // 6 - A|BC
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,-1,0,0,0,0,0,0,0,0,0,0,0,0}, // 7 - ABC
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,-1,0,0,0,0,0,0,0,0,0,0}, // 8 - CBA
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,-1,0,0,0,0,0,0,0,0}, // 9 - BAC
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,-1,0,0,0,0,0,0}, // 10 - CAB
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,-1,0,0,0,0}, // 11 - ACB
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,-1,0,0}, // 12 - BCA
};

int TRANSICAOPNRD [30] = {0,1,2,1,0,1,2,3,0,1,2,3,4,5,4,5,4,5,2,3,0,1,0,1,0,1,2,3,2,3};

int MOVIMENTO [12][22] =
{ // 0          4          9          14          19
{-1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, // 0
{ 1,-1,-1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, // 1
{ 0, 0, 1,-1,-1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, // 2
{ 0, 0, 0, 0, 1,-1,-1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, // 3
{ 0, 0, 0, 0, 0, 0, 1,-1,-1, 1, 0, 0, 0, 0, 0,-1, 1, 0, 0, 0, 0, 0, 0}, // 4
{ 0, 0, 0, 0, 0, 0, 0, 0, 1,-1,-1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, // 5
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,-1,-1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0}, // 6
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,-1, 0, 0, 0, 0, 0, 0, 0, 0, 0}, // 7
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,-1,-1, 1, 0, 0, 0, 0, 0}, // 8
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,-1,-1, 1, 0, 0}, // 9
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,-1,-1, 1}, // 10
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,-1,0,0}, // 11
};

int TRANSFORMADA [2][30] =
{
```

[illegible]

```

};

int PARPosicao[10][10][2];
// PAR ORDENADO COM POSICAO E MOVIMENTO [PROFUNDIDADE MAXIMA = 8]+1 para manipulacoes
int FILAPosicao[10][12] =
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
{-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
};

int FIMPosicao; // SE FOR = 1 A BUSCA ENCERRA
// VARIAVEIS INTERMEDIARIAS
int INTERMEDIO[10][2] ={{-1,-1},{-1,-1},{-1,-1},{-1,-1},{-1,-1},{-1,-1},{-1,-1},
{-1,-1},{-1,-1},{-1,-1}}; // VETOR COM POSICOES INICIAIS E FINAIS
int PROFUNDIDADE;
// PROFUNDIDADE DA SOLUCAO DOS ESTADOS
int REPETE = 0;
// ITERADOR PARA CONCLUIR TODOS OS MOVIMENTOS - MAIOR VALOR POSSIVEL = 9 (2*PROF+1)
int GARRA[10] = {0,1,0,1,0,1,0,1,0};
// GARRA assume valor binario - 0 aberta 1 fechada
// movimento par -> garra aberta , impar -> garra fechada
int PROFPOSICAO[10];
String STRSEND;

#include <SPI.h>
#include <Ethernet.h>
#include <Servo.h>

/**-----Informacoes a respeito da PNRD-----
// PARTE REFERENTE A PNRD
#include <Pn532NfcReader.h>
#include <PN532_HSU.h>

PN532_HSU pn532hsu(Serial1);
NfcAdapter nfc = NfcAdapter(pn532hsu);

Pn532NfcReader* reader = new Pn532NfcReader(&nfc);
Pnrd pnrd = Pnrd(reader,3,6);
/**-----**

// FUNCOES
void INICIO(), BuscaEstado(), ComparaEstado(int ColunaE,int LinhaE),
Intermediario(),BuscaPosicao(), ComparaPosicao(int ColunaP,int LinhaP);
void PrintInicial(),PrintMeio(),PrintFinal(),PrintEstado(int Est),
PrintPosicao(int POS),ENVIAR();
void VARREDURA(), CONECTA(), RECEBE(), RECEBELOCAL(), MAPEIA(), PROCESSA(), CONTROLA(),
COMPARAESTADO(), DISPARO(), ESTADOINICIAL(), ESTADOFINAL();

```

```

/**-----**
int ESTADOS[13][3][3] =
{
    { // A|B|C      Estado 0
        {0,0,0},
        {0,0,0},
        {1,2,3},
    },
    { // AB|C       Estado 1
        {0,0,0},
        {2,0,0},
        {1,0,3},
    },
    { // A|CB       Estado 2
        {0,0,0},
        {0,0,2},
        {1,0,3},
    },
    { // BA|C       Estado 3
        {0,0,0},
        {0,1,0},
        {0,2,3},
    },
    { // B|CA       Estado 4
        {0,0,0},
        {0,0,1},
        {0,2,3},
    },
    { // AC|B       Estado 5
        {0,0,0},
        {3,0,0},
        {1,2,0},
    },
    { // A|BC       Estado 6
        {0,0,0},
        {0,3,0},
        {1,2,0},
    },
    { // ABC        Estado 7
        {3,0,0},
        {2,0,0},
        {1,0,0},
    },
    { // CBA        Estado 8
        {0,0,1},
        {0,0,2},
        {0,0,3},
    },
    { // BAC        Estado 9

```

```

        {0,3,0},
        {0,1,0},
        {0,2,0},
    },
    { // CAB      Estado 10
        {0,0,2},
        {0,0,1},
        {0,0,3},
    },
    { // ACB      Estado 11
        {2,0,0},
        {3,0,0},
        {1,0,0},
    },
    { // BCA      Estado 12
        {0,1,0},
        {0,3,0},
        {0,2,0},
    },
};

//**-----**

// MAPA
String MAPA[26] =
{
    {"A150B115C55E3"}, // A (M0)->P1
    {"A150B140C75E3"}, // B (M1)->P0
    {"A150B85C34E8"}, // C (M2)->P2
    {"A150B115C55E3"}, // D (M3)->P1
    {"A150B40C28E25"}, // E (M4)->P3
    {"A150B85C34E8"}, // F (M5)->P2
    {"A90B40C28E25"}, // G (M6)->P4
    {"A150B40C28E25"}, // H (M7)->P3
    {"A90B85C34E8"}, // I (M8)->P5
    {"A90B40C28E25"}, // J (M9)->P4
    {"A90B115C55E3"}, // K (M10)->P6
    {"A90B85C34E8"}, // L (M11)->P5
    {"A90B140C75E3"}, // M (M12)->P7
    {"A90B115C55E3"}, // N (M13)->P6
    {"A30B40C28E25"}, // O (M14)->P8
    {"A90B40C28E25"}, // P (M15)->P4
    {"A30B85C34E8"}, // Q (M16)->P9
    {"A30B40C28E25"}, // R (M17)->P8
    {"A30B115C55E3"}, // S (M18)->P10
    {"A30B85C34E8"}, // T (M19)->P9
    {"A30B140C75E3"}, // U (M20)->P11
    {"A30B115C55E3"}, // V (M21)->P10
    {"G175"}, // W GARRA ABERTA
    {"G142"}, // X GARRA FECHADA
    {"A90B45C100E90"}, // Y POSICAO INICIAL
    {"A90B45C142E60"}, // Z ENCERRA ACOES

```

```

};

uint16_t* vetor1;
uint16_t* vetor2;
int inicial1[3];
int inicial2[3];
int inicial3[3];
int final1[3];
int final2[3];
int final3[3];

int estadoinicialpnrd;

int Ai, Af, Bi, Bf, Ci, Cf;

/**-----**
// Matriz PNRD

int MATRIZPNRD[13][4] =
{
    {0, 0, 0},
    {0, 1, 0},
    {0, 2, 0},
    {1, 0, 0},
    {2, 0, 0},
    {0, 0, 1},
    {0, 0, 2},
    {0, 1, 2},
    {1, 2, 0},
    {1, 0, 1},
    {2, 1, 0},
    {0, 2, 1},
    {2, 2, 0},
};

/**-----**

uint32_t tag1 = 0x12; // Bloco A
uint32_t tag2 = 0x52; // Bloco B
uint32_t tag3 = 0x72; // Bloco C
uint32_t tagId = 0xFF;
int posicao[3][3]; //Matriz do Espaco de Petri
int EstadoInicial = -1;
int EstadoFinal = -1;

int transicao;
char bloco;
bool flag = true;

```

```

bool flag1 = true;

int cont;                                // conta cliques
// PARA RECEBER OS CARACTERES
String RECEBIDO;
int LENGTH;                             // Comprimento da string
int u;                                   // Iterador na marcacao
// PARA PROCESSAR OS CARACTERES EM ACOES
String PROCESSAR, BUSCA;
//RECEBE A STRING ADVINDA DO MAPA, BUSCA OS NUMEROS ASSOCIADOS AOS MOTORES
int DIMENSAO;                            // TAMANHO DESSA STRING
int i,j;                                 // ITERADORES
// PARA EXECUTAR AS ACOES
char JUNTA[7];                           // CARACTERES QUE REPRESENTAM CADA SERVOMOTOR
bool TAREFA[7];                           // DIZ SE TEM QUE MOVER UMA CERTA JUNTA OU NAO
int ANGULO[7];                           // ANGULO A QUE DEVE SER MOVIDO O BRACO
int BRACO[7];                             // ANGULO ATUAL DO BRACO
bool DONE = false;                       // DIZ SE UMA RODADA DE MOVIMENTO ESTA ENCERRADA
Servo SERVO[7];                           // SERVOMOTORES PARA OS QUAIS OS VALORES SERAO ENVIADOS
// COM LED
int LED = 10;                             // PINO DO LED
int n = 0;
char msg[100];
char c;
int leitura = 0;
//----- ETHERNET -----
byte mac[] = {
    0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED
};
IPAddress ip(192, 168, 0, 107);
IPAddress gateway(192, 168, 0, 1);
IPAddress server(192,168,0,124);
EthernetClient client;
//-----

void setup() {
    pnrđ.setAsTagInformation(PetriNetInformation::TOKEN_VECTOR);
    pnrđ.setAsTagInformation(PetriNetInformation::ADJACENCY_LIST);
    pnrđ.setAsTagInformation(PetriNetInformation::GOAL_TOKEN);

    Serial.begin(9600);
    while (!Serial.available()){
        reader->initialize();
    }

    for (i = 0; i < 7; i++)
    {
        JUNTA[i] = 65 + i;
        TAREFA[i] = false;
    }
    INICIO();
}

```

```

}
void loop(){
    if (flag == true){
        VARREDURA();
        n = 0;
        CONECTA();
    }
    if (flag == false){
        if (EstadoInicial != -1){
            if(flag1 == true){
                RECEBE();
            }else{
                RECEBELOCAL();
            }
        }
        else{
            Serial.println("A posicao verificada nao esta cadastrada! Em 10 segundos sera
            executada uma nova varredura!");
            delay(10000);
            flag = true;
        }
    }
}

void INICIO()
{
    BRACO[0] = 90;
    BRACO[1] = 45;
    BRACO[2] = 142;
    BRACO[3] = 85;
    BRACO[4] = 60;
    BRACO[5] = 87;
    BRACO[6] = 175;
    for (i = 0; i < 7; i++) // INICIALIZA AS JUNTAS E ATRIBUI POSICAO
    {
        ANGULO[i] = BRACO[i];
        SERVO[i].write(BRACO[i]);
        delay(60);
        SERVO[i].attach(i+3); // 3 A 9
    }
}

// ----- RECEBE INFORMACOES DO SERVIDOR -----
void RECEBE()
{
    if (client.available()) {
        c = client.read();
        if(leitura == 1 && c != ','){
            msg[n] = c;
            n++;
        }
        if(c == ',') {
            leitura = 1;
        }
    }
}

```

```

        if(c == ','){
            leitura = 0;
        }
    if (!client.connected()) {
        Serial.println(msg);
        client.stop();
        n = 0;
        RECEBIDO = String(msg);
        LENGTH = RECEBIDO.length();
        MAPEIA();
    }
}
}
/**-----**
void MAPEIA()
{
    for(u=0;u<LENGTH;u++)
    {
        char PROCESSO = RECEBIDO.charAt(u);    // Caminha pela String
        int AUX = (int)PROCESSO-65;
        PROCESSAR = MAPA[AUX];
        if(AUX >= 0 && AUX < 26)
        {
            PROCESSA();
        }
        //-----Disparo das transicoes-----**
        if(AUX < 0){ //0 valor numerico recebido na string referente a transicao
            //chega neste ponto negativo por conta da conversao pela tabela Ascii, o
            //zero chega como -17.
            transicao = AUX + 17; // Para corrigir somamos 17.
            DISPARO();
            if(flag == true){ //se ocorreu execucao
                EstadoInicial = -1; //Reset do EstadoInicial (se nao tiver isso e ele nao
                // reconhecer a informacao da proxima varredura vai usar a anterior
                ANGULO[1] = 40;
                DONE = false;
                while(DONE == false){ // Esta parte vai fazer o braco subir devagar caso
                //ocorra execucao, serve pra impedir que ele esbarre em algum bloco ao
                //reiniciar a varredura
                    if (ANGULO[1] < BRACO[1])
                    {
                        BRACO[1] -= 1;
                    }
                    if (ANGULO[1] > BRACO[1])
                    {
                        BRACO[1] += 1;
                    }
                }
                SERVO[1].write(BRACO[1]);
                delay(25); //
                if (ANGULO[1] == BRACO[1]){
                    DONE = true;

```

```

        }
    }
    delay(3000);
    return;
}
}
//-----**
delay(50); // aqui vem o atraso entre movimentos
}
}
//**-----**
void PROCESSA() //
{
    DIMENSAO = PROCESSAR.length();
    for(i=0;i<7;i++) // VARRE A STRING PARA CADA JUNTA
    {
        for(j=0;j<DIMENSAO;j++)
        {
            if(PROCESSAR.charAt(j) == JUNTA[i])
            {
                TAREFA[i] = true; // existe tarefa para esse motor
                DONE = false;
                BUSCA = PROCESSAR.substring(j+1,j+4); // valor a ser enviado para o motor
                ANGULO[i] = BUSCA.toInt(); // converte em numero
                break;
                // nao existe mais de uma instancia do mesmo motor numa instrucao
            }
        }
    }
    CONTROLA();
}
//**-----**
void CONTROLA()
{
    while(DONE == false) // SE ALGUM ELEMENTO DO VETOR FOR VERDADEIRO REALIZA A OPERACAO
    {
        for(i=0;i<7;i++)
        {
            if (ANGULO[i] < BRACO[i])
            {
                BRACO[i] -= 1;
            }
            if (ANGULO[i] > BRACO[i])
            {
                BRACO[i] += 1;
            }
            if (ANGULO[i] == BRACO[i] && TAREFA[i] == true)
            {
                TAREFA[i] = false;
            }
            SERVO[i].write(BRACO[i]);
        }
    }
}

```

```

        delay(4);
    }
    if(TAREFA[0] == false && TAREFA[1] == false && TAREFA[2] == false &&
    TAREFA[3] == false && TAREFA[4] == false && TAREFA[5] == false &&
    TAREFA[6] == false)
    {
        DONE = true;
    }
}

//**-----**
void VARREDURA() //YGH CABACEJLKMKLJOQSUVTRJY
{
    Serial.println("Varredura_iniciada");
    int i, k, n;
    for(n = 0; n<10; n++){
        switch(n){
            case 0:
                i = 2; k = 0;
                RECEBIDO = "WYGH CAB"; break;
            case 1:
                i = 1; k = 0;
                RECEBIDO = "A"; break;
            case 2:
                i = 0; k = 0;
                RECEBIDO = "C"; break;
            case 3:
                i = 2; k = 1;
                Serial.println("_");
                RECEBIDO = "EJLKM"; break;
            case 4:
                i = 1; k = 1;
                RECEBIDO = "K"; break;
            case 5:
                i = 0; k = 1;
                RECEBIDO = "L"; break;
            case 6:
                Serial.println("_");
                i = 2; k = 2;
                RECEBIDO = "JOQSU"; break;
            case 7:
                i = 1; k = 2;
                RECEBIDO = "V"; break;
            case 8:
                i = 0; k = 2;
                RECEBIDO = "T"; break;
            case 9:
                Serial.println("_");
                RECEBIDO = "RJY"; break;
        }
    }
}

```

```

    LENGTH = RECEBIDO.length();
    MAPEIA();
    if(n<9){
        ReadError readError = pnrđ.getData();
        tagId = 0;
        //Caso a leitura for bem sucedida
        if(readError == ReadError::NO_ERROR){

            FireError fireError;

            //Verifica se e uma nova tag
            if(tagId != pnrđ.getTagId()){
                tagId = pnrđ.getTagId();
            }
            if(tagId == tag1) {
                tagId = 1;
                pnrđ.getGoalToken(final1);
                pnrđ.getTokenVector(inicial1);
            }
            if(tagId == tag2) {
                tagId = 2;
                pnrđ.getGoalToken(final2);
                pnrđ.getTokenVector(inicial2);
            }
            if(tagId == tag3) {
                tagId = 3;
                pnrđ.getGoalToken(final3);
                pnrđ.getTokenVector(inicial3);
            }
        }
        posicao[i][k] = tagId;
        //Serial.print(tagId, HEX);
        //Serial.print(" ");
        delay(1000);
    }
}
COMPARAESTADO();

for(n=0; n<3; n++){
    if(inicial1[n] == 1){ Ai = n;};
    if(final1[n] == 1){ Af = n;};
    if(inicial2[n] == 1){ Bi = n;};
    if(final2[n] == 1){ Bf = n;};
    if(inicial3[n] == 1){ Ci = n;};
    if(final3[n] == 1){ Cf = n;};
}
ESTADOINICIAL();
ESTADOFINAL();
Serial.print("0estadoatual:");
Serial.println(EstadoInicial);

```

```

    for(i = 0; i<3; i++){
        for(k = 0; k<3; k++){
            Serial.print(posicao[i][k], HEX);
            Serial.print(" ");
        }
        Serial.println(" ");
    }

    Serial.println("O estado inicial PNRD e:");
    Serial.println(estadoinicialpnrd);
    Serial.print("O estado desejado e:");
    Serial.println(EstadoFinal);
}

/**-----**
void COMPARAESTADO()
{
    for(n=0;n<13;n++){
        cont = 0;
        for(i=0;i<3;i++){
            for(j=0;j<3;j++){
                if(posicao[i][j] == ESTADOS[n][i][j]){
                    cont++;
                }
            }
        }
        if(cont == 9){
            EstadoInicial = n;
        }
    }
}

/**-----**
void ESTADOINICIAL()
{
    for(i=0; i<13; i++){
        if(MATRIZPNRD[i][0] == Ai){
            if(MATRIZPNRD[i][1] == Bi){
                if(MATRIZPNRD[i][2] == Ci){
                    estadoinicialpnrd = i;
                }
            }
        }
    }
}

void ESTADOFINAL()
{
    for(i=0; i<13; i++){
        if(MATRIZPNRD[i][0] == Af){
            if(MATRIZPNRD[i][1] == Bf){
                if(MATRIZPNRD[i][2] == Cf){
                    EstadoFinal = i;
                }
            }
        }
    }
}

```

```

    }
}
}
/**-----**
void DISPARO()
{
    ReadError readError = pnrđ.getData();

    //Se a leitura foi bem sucedida
    if(readError == ReadError::NO_ERROR){

        //Verifica se e uma nova tag
        if(tagId != pnrđ.getTagId()){
            tagId = pnrđ.getTagId();
        }
        if(tagId == tag1) {
            bloco = 'A';
        }
        if(tagId == tag2) {
            bloco = 'B';
        }
        if(tagId == tag3) {
            bloco = 'C';
        }

        //Dispara a transicao
        FireError fireError = pnrđ.fire(transicao);

        switch (fireError){
            case FireError::NO_ERROR :
                //Atualizando a tag
                if(pnrđ.saveData() == WriteError::NO_ERROR){
                    Serial.print("0bloco");
                    Serial.print(bloco);
                    Serial.println("foi atualizado com sucesso.");
                    return;
                }else{
                    Serial.println("Erro na atualizacao da tag");
                    return;
                }

            case FireError::PRODUCE_EXCEPTION :
                Serial.println("Erro: Excecao gerada. 0bloco seguiu
pelo caminho errado.");
                flag = true; //caso tiver excecao o processo deve ser reiniciado.
                return;

            case FireError::CONDITIONS_ARE_NOT_APPLIED :
                return;
        }
    }
}

```

```

    }
}

void CONECTA()
{
    // ----- Conexao -----
    if (Ethernet.begin(mac) == 0) {
        Ethernet.begin(mac, ip);
    }
    Serial.println("Conectando...");
    if (client.connect(server, 80)) {
        Serial.println("Conectado!");
        client.print("GET http://192.168.0.124/index.php?i=");
        client.print(EstadoInicial);
        client.print("&f=");
        client.print(EstadoFinal);
        client.println(" HTTP/1.1");
        client.println("Host: 192.168.0.124");
        client.println("Connection: close");
        client.println();
    }
    else {
        Serial.println("Falha ao conectar. Alternando para planejamento local");
        flag1 = false;
    }
}

// -----
}

void RECEBELOCAL()
{
    BuscaEstado(); // FUNCAO PARA NAVEGAR PELOS ESTADOS

    PrintMeio(); // SOLUCAO A NIVEL DE ESTADOS

    Intermediario(); // USA A TRANSFORMADA PARA GERAR O VETOR DE POSICOES DESEJADAS

    BuscaPosicao(); // REALIZA AS BUSCAS DE POSICAO

    PrintFinal(); // FUNCAO PARA MOSTRAR A SOLUCAO FINAL EM POSICAO

    ENVIAR(); // MOSTRA QUAL A SEQUENCIA DE CARACTERES A SEREM PROCESSADOS
    //EM MOVIMENTO NO MICRO
}

// -----
void BuscaEstado()
{
    int i,j; // ITERADORES
    while(FIMEstado == 0)
    {
        for(i=0;i<30;i++)
        {
            if(TRANSICOES[EstadoAtual][i] == -1) // sai do estado atual
            {

```

```

        for(j=0; j<13; j++)
        {
            if(TRANSICOES[j][i] == 1) // vai para o novo estado
            {
                ComparaEstado(j,i);
            }
        }
        if(FIMEstado == 1)
        {
            break;
        }
    }
}

if(FIMEstado == 0) // No atual recebe primeiro elemento da fila
{
    EstadoAtual = FILAEstado[0];
}

for(i=0; i<12; i++) // Fila se atualiza, jogando elementos para frente (pushforward)
{
    FILAEstado[i] = FILAEstado[i+1];
}
FILAEstado[12] = -1;
}

}

//-----
void ComparaEstado(int ColunaE, int LinhaE) // COMPARA AO ESTADO FINAL
//ALEM DE ESTABELECEER UMA FILA CASO NAO SEJA O ESTADO FINAL
{
    int k; // ITERADOR DA FUNCAO
    if(EstadoTrack[ColunaE] == -1) // CASO AINDA NAO TENHA UM PAI, RECEBE UM
    {
        EstadoTrack[ColunaE] = EstadoAtual;
        TransicaoTrack[ColunaE] = LinhaE;
    }
    if(EstadoAtual == EstadoFinal) // ENCERRA A BUSCA EM ESTADOS
    {
        FIMEstado = 1;
    }
    else // ADICIONA NA FILA DE ESTADOS
    {
        for(k=0; k<13; k++)
        {
            if(ColunaE == FILAEstado[k]) // CASO O ELEMENTO EXISTA NA FILA ELE
//NAO E ADICIONADO NOVAMENTE
            {
                break;
            }
        }
        if(FILAEstado[k] == -1) // COLOCA ELE NA ULTIMA POSICAO DA FILA E QUEBRA
        {
            FILAEstado[k] = ColunaE;
            break;
        }
    }
}

```

```

    }
}
}

//-----
void PrintMeio() // MOSTRA A SOLUCAO A NIVEL DE ESTADOS E DA OS PARAMETROS
//PARA A BUSCA EM POSICAO
{
    int q,u,l,p=5,contador=5; // comparador dessa funcao e iteradores
    q = EstadoFinal;
    //Serial.print("\n\n");
    while(q != EstadoInicial) // transfere dados de transicao e pai para o vetor ORDEM
    {
        PAREstado[p][0] = q;
        PAREstado[p][1] = TransicaoTrack[q];
        q = EstadoTrack[q]; // VAI PARA O PAI
        p--;
    }
    PAREstado[p][0] = q; // recebe estado inicial
    PAREstado[p][1] = TransicaoTrack[q];
    while(PAREstado[0][0] == -1)
    {
        for(p = 0; p <(6-1) ; p++) // pushforward ate que o primeiro elemento esteja
        //ocupado - 5 pois e a quantidade de linhas de PAREstado
        {
            PAREstado[p][0] = PAREstado[p+1][0];
            PAREstado[p][1] = PAREstado[p+1][1];
        }
        contador--;
    }
    while(1)
    {
        for(p = 0; p < 6; p++)
        {
            PAREstado[p][1] = PAREstado[p+1][1];
        }
        if(PAREstado[0][1] != -1)
        {
            break;
        }
    }
    PROFUNDIDADE = contador; // aqui marca a profundidade da solucao
}

//-----
void Intermediario()
{
    int i,j,k; // ITERADORES
    int u = 0; // contador para dizer estado da GARRA
    // PREDEFINICAO: ESTADO INICIAL E FINAL DA GARRA DEVERA SER 4
    INTERMEDIO[u][0] = 4;
    INTERMEDIO[u][1] = TRANSFORMADA[0][PAREstado[0][1]];
}

```

```

u++;
for(k = 0; k < PROFUNDIDADE; k++)
{
    INTERMEDIO[u][0] = TRANSFORMADA[0][PAREstado[k][1]];
    // POSICAO INICIAL DA ITERACAO K
    INTERMEDIO[u][1] = TRANSFORMADA[1][PAREstado[k][1]];
    // POSICAO FINAL DA ITERACAO K
    u++;
    // caso impar
    if(k < PROFUNDIDADE-1)
    {
        INTERMEDIO[u][0] = TRANSFORMADA[1][PAREstado[k][1]];
        // TRANSICAO DE GARRA ABERTA
        INTERMEDIO[u][1] = TRANSFORMADA[0][PAREstado[k+1][1]];
        u++;
    }
}

INTERMEDIO[u][0] = TRANSFORMADA[1][PAREstado[k][1]];
// POSICAO INICIAL DA ITERACAO K
INTERMEDIO[u][1] = 4;
// POSICAO FINAL DA ITERACAO K
for(i=0;i<10;i++)
{
    for(j=0;j<10;j++)
    {
        for(k=0;k<2;k++)
        {
            PARPosicao[i][j][k] = -1;
        }
    }
}

}

//-----
void BuscaPosicao()
{
    int i,j;
    while(REPETE < (PROFUNDIDADE*2)+1)
    // Executa todas as movimentacoes necessarias
    {
        PosicaoInicial[REPETE] = INTERMEDIO[REPETE][0];
        PosicaoFinal[REPETE] = INTERMEDIO[REPETE][1];
        PosicaoAtual[REPETE] = PosicaoInicial[REPETE];
        while(FIMPosicao == 0)
        {
            for(i=0;i<22;i++)
            {
                if(MOVIMENTO[PosicaoAtual[REPETE]][i] == -1)
                // sai da posicao atual
                {
                    for(j=0;j<12;j++)
                    {

```

```

        if(MOVIMENTO[j][i] == 1)
        // vai para a nova posicao
        {
            ComparaPosicao(j,i);
        }
    }
    if(FIMPosicao == 1)
    {
        break;
    }
}
if(FIMPosicao == 0)
{
    PosicaoAtual[REPETE] = FILAPosicao[REPETE][0];
}
for(i=0;i<12;i++)
// Fila se atualiza, jogando elementos para frente (pushforward)
{
    FILAPosicao[REPETE][i] = FILAPosicao[REPETE][i+1];
}
    FILAPosicao[REPETE][11] = -1;
}
    REPETE++;
    FIMPosicao = 0;
}
}
//-----
void ComparaPosicao(int ColunaP,int LinhaP)
{
    int k; // ITERADOR DA FUNCAO
    if(PosicaoTrack[REPETE][ColunaP] == -1)
    // CASO AINDA NAO TENHA UM PAI, RECEBE UM
    {
        PosicaoTrack[REPETE][ColunaP] = PosicaoAtual[REPETE];
        MovimentoTrack[REPETE][ColunaP] = LinhaP;
    }
    if(PosicaoAtual[REPETE] == PosicaoFinal[REPETE])
    // ENCERRA A BUSCA EM ESTADOS
    {
        FIMPosicao = 1;
    }
    else // ADICIONA NA FILA DE ESTADOS
    {
        for(k=0;k<12;k++)
        {
            if(ColunaP == FILAPosicao[REPETE][k])
// CASO O ELEMENTO EXISTA NA FILA ELE NAO E ADICIONADO NOVAMENTE
            {
                break;
            }

```

```

        if(FILAPosicao[REPETE][k] == -1)
        // COLOCA ELE NA ULTIMA POSICAO DA FILA E QUEBRA
        {
            FILAPosicao[REPETE][k] = ColunaP;
            break;
        }
    }
}

//-----
void PrintFinal()
{
    REPETE = 0;
    while(REPETE < ((PROFUNDIDADE*2)+1))
    {
        int q,u,l,p=8,contador=8; // comparador dessa funcao e iteradores
        q = PosicaoFinal[REPETE];
        while(q != PosicaoInicial[REPETE])
        // transfere dados de transicao e pai para o vetor ORDEM
        {
            PARPosicao[REPETE][p][0] = q;
            PARPosicao[REPETE][p][1] = MovimentoTrack[REPETE][q];
            q = PosicaoTrack[REPETE][q]; // VAI PARA O PAI
            p--;
        }
        PARPosicao[REPETE][p][0] = q; // recebe posicao inicial
        PARPosicao[REPETE][p][1] = MovimentoTrack[REPETE][q];
        // recebe movimento lixo
        while(PARPosicao[REPETE][0][0] == -1)
        {
            for(p = 0; p < (10-1) ; p++)
            // pushforward ate que o primeiro elemento esteja ocupado - 10
            //pois e a quantidade de linhas de PARPosicao
            {
                PARPosicao[REPETE][p][0] = PARPosicao[REPETE][p+1][0];
                PARPosicao[REPETE][p][1] = PARPosicao[REPETE][p+1][1];
            }
            contador--;
            PARPosicao[REPETE][p][1] = PARPosicao[REPETE][p+1][1];
        }
        while(1)
        {
            for(p = 0; p < 10; p++)
            {
                PARPosicao[REPETE][p][1] = PARPosicao[REPETE][p+1][1];
            }
            if(PARPosicao[REPETE][0][1] != -1)
            {
                break;
            }
        }
    }
}

```

```
    PROFPOSICAO[REPETE] = contador;
    // aqui marca a profundidade da solucao
    REPETE++;
}
}
//-----
void ENVIAR() // ENVIA A STRING DE COMANDOS QUE O BRACO DEVE EXECUTAR
{
    int i;
    REPETE = 0;
    while(REPETE < ((PROFUNDIDADE*2) + 1))
    {
        char AUX1 = (87+(REPETE%2));
        STRSEND.concat(AUX1);
        if ((REPETE/2) >0 && (REPETE%2) == 0){
            STRSEND.concat(TRANSICAOPNRD[PAREstado[(REPETE-1)/2][1]]);
        }
        for(i=0;i<PROFPOSICAO[REPETE];i++)
        {
            char AUX1 = (65+PARPosicao[REPETE][i][1]);
            STRSEND.concat(AUX1);
        }
        REPETE++;
    }
    RECEBIDO = STRSEND;
    LENGTH = RECEBIDO.length();
    Serial.print("\n\t");
    Serial.println(RECEBIDO);
    MAPEIA();
}
```

APÊNDICE C – ELEMENTOS DO MODELO IPNRD

Estados:

- BAC - Bloco B sobre A sobre C
- AC_B - Bloco A sobre C e B em coluna distinta
- AB_C - Bloco A sobre B e C em coluna distinta
- CAB - Bloco C sobre A sobre B
- BCA - Bloco B sobre C sobre A
- CA_B - Bloco C sobre A e B em coluna distinta
- A_B_C - A, B e C em colunas distintas
- BA_C - Bloco B sobre A e C em coluna distinta
- CBA - Bloco C sobre B sobre A
- CB_A - Bloco C sobre B e A em coluna distinta
- BC_A - Bloco B sobre C e A em coluna distinta
- ACB - Bloco A sobre C sobre B
- ABC - Bloco A sobre B sobre C

Transições:

- m0A2T - Movimento do bloco A para cima da mesa
- m0A2B - Movimento do bloco A para cima de B
- m0A2C - Movimento do bloco A para cima de C
- m1A2T - Movimento do bloco A para cima da mesa
- m1C2T - Movimento do bloco C para cima da mesa
- m0B2A - Movimento do bloco B para cima de A
- m0C2A - Movimento do bloco C para cima de A
- m0B2T - Movimento do bloco B para cima da mesa

- m0C2T - Movimento do bloco C para cima da mesa
- m0B2C - Movimento do bloco B para cima de C
- m0C2B - Movimento do bloco C para cima de B
- m1B2T - Movimento do bloco B para cima da mesa
- m2B2C - Movimento do bloco B para cima de C
- m3B2T - Movimento do bloco B para cima da mesa
- m1C2A - Movimento do bloco C para cima de A
- m1C2B - Movimento do bloco C para cima de B
- m2A2C - Movimento do bloco A para cima de C
- m3A2T - Movimento do bloco A para cima da mesa
- m2A2B - Movimento do bloco A para cima de B
- m2A2T - Movimento do bloco A para cima da mesa
- m1B2C - Movimento do bloco B para cima de C
- m1B2A - Movimento do bloco B para cima de A
- m3C2T - Movimento do bloco C para cima da mesa
- m2C2B - Movimento do bloco C para cima de B
- m2C2T - Movimento do bloco C para cima da mesa
- m2C2A - Movimento do bloco C para cima de A
- m1A2C - Movimento do bloco A para cima de C
- m1A2B - Movimento do bloco A para cima de B
- m2B2T - Movimento do bloco B para cima da mesa
- m2B2A - Movimento do bloco B para cima de A

APÊNDICE D – ELEMENTOS DO MODELO PNRD

Estados:

- AB - Bloco A sobre o bloco B
- AC - Bloco A sobre o bloco C
- AT - Bloco A sobre a mesa

Transições:

- TA1 - Movimento do bloco A para cima de C
- TA2 - Movimento do bloco A para cima da mesa
- TA3 - Movimento do bloco A para cima de B
- TA4 - Movimento do bloco A para cima da mesa
- TA5 - Movimento do bloco A para cima de B
- TA6 - Movimento do bloco A para cima de C

APÊNDICE E – DESCRIÇÃO DAS POSIÇÕES REFERENTES A CADA LETRA DA MENSAGEM ENVIADA AO SISTEMA LOCAL

Cada letra corresponde a uma combinação de angulação das juntas do braço robótico. Existem quatro juntas: A, B, C, E e G.

- A - A150B115C55E3
- B - A150B140C75E3
- C - A150B85C34E8
- D - A150B115C55E
- E - A150B40C28E25
- F - A150B85C34E8
- G - A90B40C28E25
- H - A150B40C28E25
- I - A90B85C34E8
- J - A90B40C28E25
- K - A90B115C55E3
- L - A90B85C34E8
- M - A90B140C75E3
- N - A90B115C55E3
- O - A30B40C28E25
- P - A90B40C28E25
- Q - A30B85C34E8
- R - A30B40C28E25
- S - A30B115C55E3
- T - A30B85C34E8

- U - A30B140C75E3
- V - A30B115C55E3
- W - G175
- X - G142
- Y - A90B45C100E90