

LUCAS HENRIQUES SILVA

**PROJETO E IMPLEMENTAÇÃO DE PNRD E PNRD
ESTENDIDA A DISTÂNCIA**



UNIVERSIDADE FEDERAL DE UBERLÂNDIA

FACULDADE DE ENGENHARIA MECÂNICA

2017

LUCAS HENRIQUES SILVA

**PROJETO E IMPLEMENTAÇÃO DE PNRD E PNRD ESTENDIDA A
DISTÂNCIA**

Monografia de Conclusão de Curso
apresentada ao programa de Graduação em
Engenharia Mecatrônica da Universidade
Federal de Uberlândia como parte dos requisitos
para obtenção do título de **Bacharel em**
Engenharia Mecatrônica.

Área de Concentração: Automação e Robótica.

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

UBERLÂNDIA

2017

Agradecimentos

Aos meus pais e familiares, pelo carinho, amor, compreensão, incentivo, patrocínio, e por estarem presentes nos vários momentos de alegria vividos, e também nos momentos de dificuldade.

À Universidade Federal de Uberlândia e à Faculdade de Engenharia Mecânica pela oportunidade de realizar esse curso.

Aos amigos que fiz durante a graduação, principalmente aos que fazem parte da 14ª Turma de Engenharia Mecatrônica, aos grandes amigos de infância que me acompanham até os dias de hoje.

Ao professor e orientador Dr. Jean Tavares, por se mostrar sempre disponível a dar orientações e críticas, sempre visando ao meu crescimento profissional e também pessoal e por também serem exemplos de amor à profissão a qual almejo.

Aos companheiros presentes no *MAPL (Manufacturing Automated Planning Lab)*, agradeço a amizade e a disposição a ajudar sempre que eu precisava.

SILVA, L.H. **PROJETO E IMPLEMENTAÇÃO DE PNRD E PNRD ESTENDIDA A DISTÂNCIA**. 2017. Monografia de Conclusão de Curso (Bacharel) – Curso de Graduação em Engenharia Mecatrônica, Universidade Federal de Uberlândia, Uberlândia, 2017.

RESUMO

A tecnologia RFID (*Radio Frequency Identification*) é utilizada cada vez mais em aplicações industriais e logísticas, porém, apenas de maneira a permitir a identificação de itens. Neste trabalho, implementa-se um modelo baseado nas redes de Petri proposto por [1] em que possibilita, além da identificação do produto, realizar uma atualização do estado do item com relação ao processo esperado em tempo real. A rede de Petri em questão foi denominada PNRD (*Petri Net Inside RFID Database* ou Rede de Petri Inserida em Base de Dados RFID) e define que as etiquetas RFID são usadas como um base de dados contendo ambos a identificação do item, processo esperado (matriz de incidência da rede de Petri) e estado atual do item, e no *middleware* estão as informações sobre como o processo progride em função das pré-condições de captura dos dados das etiquetas (antena, distância, estado da etiqueta) necessárias para o disparo das transições da rede de Petri. Esse trabalho apresenta o *middleware* implementado para obter as informações das etiquetas e armazenar as informações de disparo das transições. Um exemplo de implementação mostra o controle unidirecional de um veículo utilizando a PNRD.

Palavras Chave: RFID, PNRD, Rede de Petri, DEMIS, RETIM.

SILVA, L.H. **PROJECT AND IMPLEMENTATION OF PNRD AND EXTENDED PNRD THROUGH DISTANCE**. 2017. Final Course Assignment (Bachelor) – Mechatronics Engineer Graduation, Universidade Federal de Uberlândia, Uberlândia, 2017.

ABSTRACT

RFID (Radio Frequency Identification) technology is being used increasingly on industrial and logistics applications, however, just as a manner of enabling item identification. In this work, a model based on Petri Nets, proposed by [2] is implemented, that, besides the identification of the product, it allows to update the item state in relation to the expected process in real-time. The Petri Net in question was named PNRD (Petri Net Inside RFID Database) and defines that RFID tags are used as a database containing the item identification, the expected process that said item will go through (stored on the Incident Matrix from the PNRD) and the current state of the item, besides that, in the middleware, are the information about how the process advances in function of pre-conditions determined from reads from the tags (antenna, distance, state of tag) needed to trigger the Petri Net's transition. This work presents the implemented middleware to extract information from tags and store the trigger information from transitions. As well an example of implementation that shows a unidirectional control of a vehicle using PNRD.

Keywords: RFID, PNRD, Petri Nets, DEMIS, RETIM.

Lista de Figuras

Figura 1: Esquema de um sistema RFID [4]	7
Figura 2: Frequências do padrão RFID [5]	9
Figura 3: Tag Passiva (menor) ao lado de uma Tag Ativa (maior) [5]	11
Figura 4: Estrutura de organização dos bancos de memória da tag utilizada [8]	11
Figura 5: Antena Acura utilizada [4]	12
Figura 6: Radiação Antena Direcional [5]	13
Figura 7: Radiação Antena Omnidirecional [5]	13
Figura 8: Feixes de antenas direcionais de diferentes ganhos. [5]	13
Figura 9: Polarização linear x polarizações circulares. [5]	14
Figura 10: Rede de Petri elementar e sua matriz incidência [2]	16
Figura 11: Exemplo de vetorização de uma matriz incidência	17
Figura 12: Representação em rede de Petri de sistema de entregas. Adaptada de [2]....	18
Figura 13: Nova marcação na rede de Petri de sistema de entregas. Adaptada de [2]... 18	
Figura 14: Fluxograma DEMIS (Fonte: Próprio Autor)	21
Figura 15: Exemplo de arquivo de parametrização (Fonte: Próprio autor)	23
Figura 16: Representação gráfica pré-modificações RETIM (Fonte: Próprio autor)	24
Figura 17: Representação gráfica pós-modificações RETIM (Fonte: Próprio autor) ...	25
Figura 18: Informações das tags no estado 00 (Fonte: Próprio autor)	25
Figura 19: Estrutura de testes para PNRD	26
Figura 20: Tag UHF utilizada	27
Figura 21: Montagem do caso de uso. Adaptado de [14]	28
Figura 22: Arquivo de configuração usado. Adaptado de [14]	29
Figura 23: PNRD estendida a distância referente ao controle do Robô Móvel. [14]	30
Figura 24: Exemplo de Transição T2 sendo disparada, depois de decisão. [14]	31

Lista de Tabelas

Tabela 1: Isolamento de cabos comumente usados e suas perdas [5]	14
Tabela 2: Estrutura de organização da memória EPC	22

Sumário

Lista de Figuras	iv
Sumário	v
CAPÍTULO I.....	6
1.1. Objetivos.....	7
1.2. Justificativa	8
CAPÍTULO II.....	9
2.1. Sistema RFID.....	9
2.1.1. <i>Tag</i> RFID.....	10
2.1.2. Antena RFID	12
2.1.3. Leitor RFID	15
2.2. Redes de Petri	15
2.3. PNRD.....	16
2.4. PNRD Estendida a Distância	19
CAPÍTULO III	20
3.1. Introdução	20
3.2. DEMIS	20
3.3. RETIM	23
CAPÍTULO IV	26
4.1. Introdução	26
4.2. Casos de uso	28
CAPÍTULO V	32
REFERÊNCIAS BIBLIOGRÁFICAS	33
APÊNDICE A – DEMIS CORE	35
APÊNDICE B – DEMIS – IncidentMatrix	51
APÊNDICE C – DEMIS – ReaderWord.....	57
APÊNDICE D – DEMIS – StatesVector.....	59
APÊNDICE E – DEMIS – TagRead	62
APÊNDICE F – RETIM – Visualizacao_Petri.....	65
APÊNDICE G – RETIM – Resolve_Matriz	80

CAPÍTULO I

INTRODUÇÃO

RFID é a denominação genérica para sistemas compostos de dispositivos eletrônicos que permitem realizar identificações utilizando comunicação por RF (radiofrequência) entre os elementos básicos que compõem este sistema: o equipamento interrogador ou leitor (*reader*) e as *tags* (etiquetas eletrônicas de identificação)[2].

Ainda hoje a RFID é vista apenas como uma ferramenta substituta do código de barras, mas essa tecnologia se torna crescentemente importante e é utilizada em produção, manufatura, e cadeia de suprimentos. Muitas aplicações de RFID são cenários fechados desenvolvidos para resolver problemas particulares na indústria quando soluções alternativas não são possíveis.

As ferramentas RFID têm assumido um papel importante para auxiliar linhas de montagem e processos de cadeia de suprimentos e logísticos, visto que, ao contrário do código de barras, é possível ler e alterar dinamicamente as informações contidas nas tags. Essas ferramentas também conseguem identificar, categorizar e gerenciar fluxo de itens e informação através da cadeia de suprimentos, dando maior visibilidade ao processo de negócio e garantindo a transferência de dados adequada para atingir as condições ótimas da cadeia[3].

Todo sistema RFID é composto de, na maioria das vezes, 4 entidades: a etiqueta ou *tag* RFID, antena, leitor e *middleware*, como ilustra a Figura 1. Este trabalho foca-se na construção do *middleware*, responsável por controlar o leitor e servir de camada de comunicação com softwares.

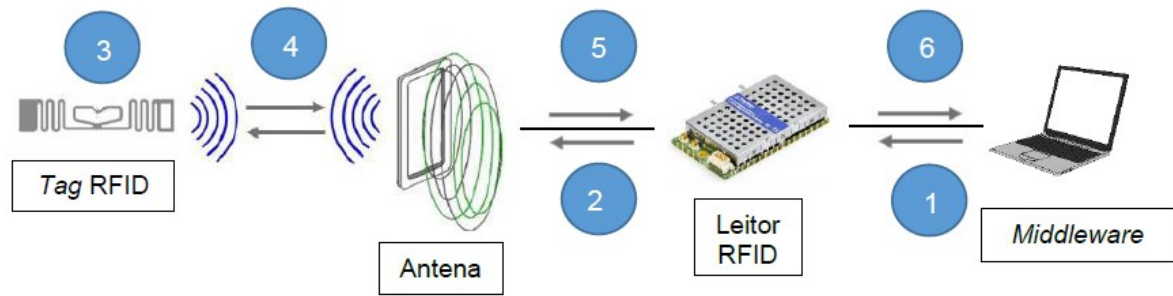


Figura 1: Esquema de um sistema RFID [4]

Este trabalho propõe que a informação siga uma forma de controle proposta por [1], o que muda o paradigma de como se é utilizada a tecnologia RFID atualmente, acrescentando maior grau de controle sobre os itens etiquetados, pois descentraliza a informação. Grava-se além da informação sobre o item em sua respectiva *tag*, armazena-se também a informação sobre o processo que o mesmo passa, já as informações quanto à mudança de estado do processo, ficam a cargo do *middleware*.

1.1.Objetivos

Assim sendo, fica claro que é definido como objetivo desenvolver um software que siga a orientação de [1]. Tem-se então:

Objetivo Principal: Implementar o algoritmo de PNRD e PNRD estendida a distância em C++ ou Java.

Objetivo Específicos:

- Ampliar os conhecimentos em programação;
- Estudar os Princípios básicos de funcionamento do RFID.
- Estudar os algoritmos de PNRD e PNRD estendida;
- Implementar o algoritmo de PNRD;
- Implementar o algoritmo de PNRD estendida a distância;
- Realizar testes;

1.2. Justificativa

A tecnologia RFID continua a se desenvolver nos meios da produção industrial e logística. Desta forma, um modelo que possa permitir o controle mais acurado do produto e seu respectivo processo, é justificável, podendo gerar mais valor ao processo produtivo e logístico na forma de redução de perdas e monitoramento e, em alguns casos, efetuar o tratamento de erros de forma automática.

A PNRD vai além dos benefícios da tecnologia RFID, pois permite a utilização de informações do processo (usando-se de redes de Petri) e, usando da tecnologia RFID como sensor, é possível usar a própria leitura realizada pelo sistema para determinar a distância entre a tag RFID e o leitor [4]. Pode-se usar então desta informação para o funcionamento da PNRD, esta versão da PNRD é denominada como PNRD estendida a distância.

CAPÍTULO II

FUNDAMENTAÇÃO TEÓRICA

Este capítulo trata de assuntos pertinentes à compreensão e ao desenvolvimento do trabalho; dentre eles, princípios básicos de funcionamento de um sistema RFID, redes de Petri e PNRD.

2.1. Sistema RFID

Todo sistema RFID opera em determinada frequência, existindo três padrões disponíveis, como ilustra a Figura 2.

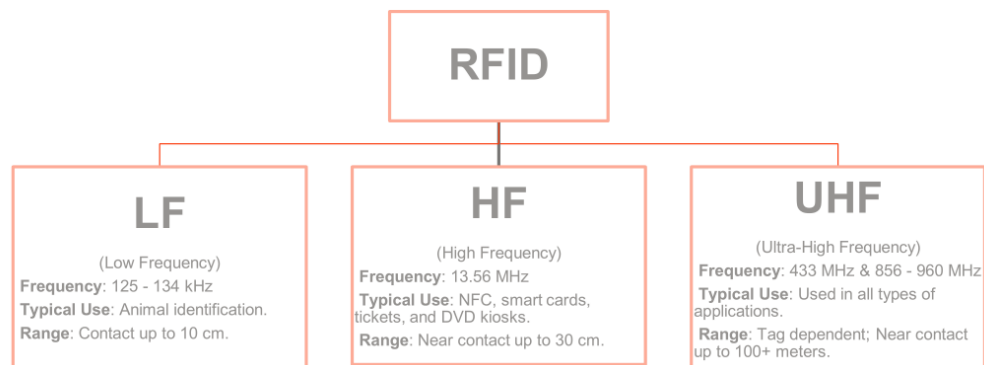


Figura 2: Frequências do padrão RFID [5]

Há três frequências principais dos sistemas RFID, LF (baixa frequência), HF (alta frequência) e UHF (ultra alta frequência). Cada uma tem um intervalo determinado de frequência, uso típico e distância máxima de operação. A distância máxima de operação do sistema cresce proporcionalmente à frequência empregada, já que uma onda de maior frequência é capaz de transmitir mais energia, sendo assim um sistema LF apresenta uma distância de operação muito menor que um sistema UHF.

Porém, frequências maiores implicam em maiores interferências advindas de materiais metálicos ou de constante dielétrica elevada (líquidos em geral). Deste modo, antenas e *tags* operando em ambientes contendo objetos metálicos, que apresentam alto índice de reflexão das ondas, podem sofrer perdas significativas se não tratados corretamente.

Como já foi dito, o sistema RFID é composto por 4 partes, sendo 3 delas físicas: a tag, o leitor e antena RFID [6]. Serão mostrados os princípios de funcionamentos dos mesmos em seguida.

2.1.1. Tag RFID

As *tags* RFID funcionam como transponders, ou seja, recebem informação pré-determinada (no caso via radiofrequência) e retransmitem um sinal, também pré-determinado. São compostas de microchips de silício, onde é feito o processamento da mensagem recebida e da mensagem a ser enviada, além de conter também a memória, onde fica armazenada as informações referentes à *tag*, e composta também de antena(s) para receber o sinal RF e responder adequadamente.

Tags RFID podem ser divididas em duas principais categorias, ativas e passivas. Sendo as *tags* passivas não necessitando de fonte de energia para responder as mensagens recebidas, pois o próprio sinal da mensagem é usado para energiza-las, o que torna a distância de leitura dessas *tags* muito baixa (no máximo 10 metros[7]).

Já as *tags* RFID ativas possuem alimentação, e usam desta para gerar o sinal de resposta. Deste modo, seu sinal de resposta, muitas vezes apresenta maior potência do que o de uma tag passiva. Além disso, esse tipo de tag pode enviar um sinal próprio de maneira contínua. A Figura 3 ilustra uma comparação de uma tag ativa acima de uma tag passiva.



Figura 3: Tag Passiva (menor) ao lado de uma Tag Ativa (maior) [5]

Existe também um tipo de *tag* híbrida, chamado de semipassiva, que o princípio de funcionamento é basicamente o mesmo da tag passiva, porém com uma bateria interna, permitindo assim, distâncias de comunicação mais longas.

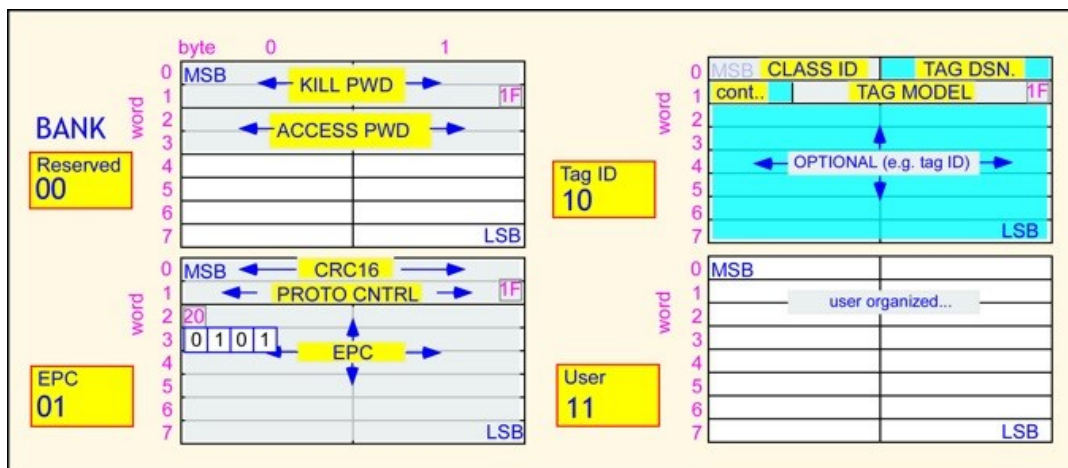


Figura 4: Estrutura de organização dos bancos de memória da tag utilizada [8]

Como ilustra a Figura 4, as tags do tipo GEN2 (utilizadas neste trabalho), possuem, em sua maioria, quatro bancos de memória com a seguinte estrutura:

O banco reservado, contém os dados de controle de acesso da tag às quatro primeiras palavras (oito primeiros bytes) destinadas a senha utilizada para inutilizar a tag e as quatro seguintes destinadas a senha de acesso a tag.

Quanto ao banco EPC, o mesmo sempre é transmitido em todas as comunicações, e as duas primeiras palavras da mensagem, sendo a primeira para controle de erro, e a segunda para controle de protocolo (metadados da mensagem).

O banco reservado para *Tag Id* é somente leitura e contém os metadados da tag, como modelo, fabricante, tipo de tag, frequência de operação e identificação da mesma.

O banco de Usuário é basicamente um banco de memória, com capacidades de leitura e escrita, designado para inserir qualquer tipo de informação. Este banco pode, ou não, estar presente na tag, dependendo do modelo.

2.1.2. Antena RFID

A antena serve de ponte entre as *tags* e o leitor, realizando a modulação dos sinais de radiofrequência enviados pelo leitor, e a demodulação dos sinais recebidos das tags. A Figura 5 apresenta uma antena monoestática, ou seja, a mesma antena realiza as operações de leitura e escrita nas tags.



Figura 5: Antena Acura utilizada [4]

As antenas são classificadas quanto à diretividade, ganho, polarização e frequência de operação. Quanto à diretividade a antena, esta pode ser direcional, irradiando para apenas uma

direção em que a antena esteja apontada, como na Figura 6, ou omni-direcional, em que irradia para todas as direções, como na Figura 7.

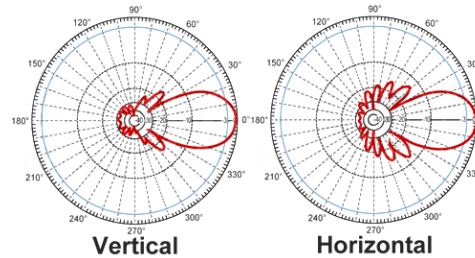


Figura 6: Radiação Antena Direcional [5]

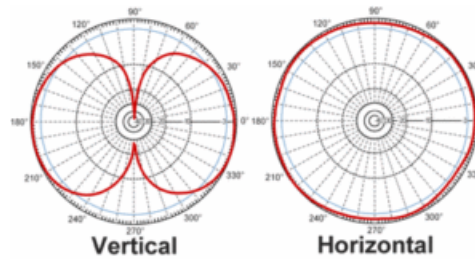


Figura 7: Radiação Antena Omnidirecional [5]

O ganho determina quantas vezes a antena ampliará o sinal do leitor, é dado em decibéis (dBi) e é atrelado à largura do feixe emitido por uma antena direcional, como ilustra a Figura 8. Sendo que uma antena de menor ganho implica numa maior largura de feixe, e vice-versa.

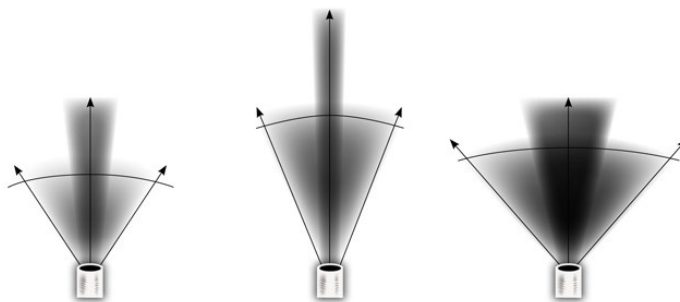


Figura 8: Feixes de antenas direcionais de diferentes ganhos. [5]

A polarização da antena define como o sinal RF viaja pelo meio depois de enviado pela antena, podendo esta ser linear ou circular, como mostra a Figura 9.

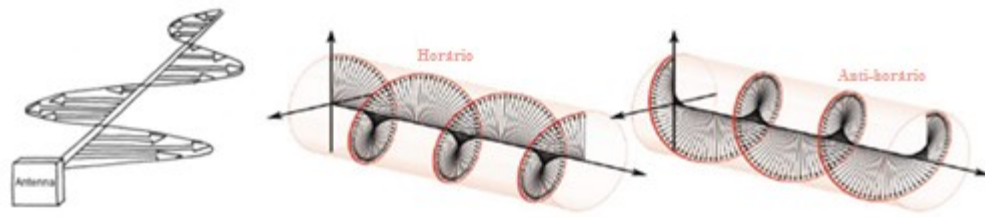


Figura 9: Polarização linear x polarizações circulares. [5]

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

Já antenas de polarização circular, emitem seu sinal de maneira helicoidal, podendo ser no sentido horário ou anti-horário, como ilustra a Figura 9. Desta maneira, transmitindo o sinal em vários planos simultaneamente, o que permite que a leitura das *tags* seja feita em diversas orientações, porém, ao custo da distância de leitura ser menor que sua contraparte. A direção da polarização (horário ou anti-horário) é importante de se considerar ao usar múltiplas antenas, pois antenas de mesma polaridade não podem transmitir ao mesmo tempo por gerarem interferências destrutivas, problema que não ocorre ao se usar múltiplas antenas conectadas a apenas um leitor, pois o mesmo consegue administrar as antenas de maneira que não transmitam ao mesmo tempo [5].

Deve-se levar em consideração também a conexão entre a antena e o leitor, realizada por um cabo coaxial que gera também uma perda em dB em função do comprimento do cabo e seu isolamento. São dados quatro exemplos de cabos com diferentes isolamentos e, consequentemente, diferentes perdas na tabela 1.

Tabela 1: Isolamento de cabos comumente usados e suas perdas [5]

Cabo	LMR-195	LMR-240	LMR-400	LMR-600

Perda/30m (dB)	11.1	7.6	3.9	2.5
-----------------------	------	-----	-----	-----

2.1.3. Leitor RFID

O leitor é responsável por coordenar as operações de leitura e escrita e enviá-las para a antena de maneira a serem executadas de acordo com o protocolo da *tag* (neste caso GEN2).

O leitor RFID é o *hardware* que controla diretamente a antena, de maneira a modular os comandos de leitura e escrita a serem enviados para as *tags* via RF, e demodular as respostas recebidas pela antena advindas da *tag*[9].

Os leitores RFID variam quanto ao número de antenas suportadas simultaneamente, método de comunicação, a frequência de operação, a energia de transmissão (consequentemente a distância máxima de leitura), velocidade (número de leituras por segundo) e o tipo de conexão (em geral, USB, WI-FI ou serial).

2.2. Redes de Petri

Rede de Petri é uma técnica de modelagem que permite a representação de sistemas, de forte base matemática. Essa técnica possui a particularidade de permitir modelar sistemas paralelos, concorrentes, assíncronos e não-determinísticos[10].

Os lugares representam os estados possíveis do sistema que um *token* ou marcação pode ocupar, os *tokens*, ou marcações representam qual estado se encontra determinado elemento. O arco serve para representar de e para quais estados podem disparar determinada transição. O disparo é dado pela ativação de uma transição, removendo uma marcação de um lugar e adicionando a outro. Sendo assim, no exemplo do caso da Figura 10, existe uma marcação no Estado 0, que quando disparada remove uma marcação da Transição 0 e a insere no Estado 1.

Apresentam representações gráficas padronizadas, além de uma representação matricial. Sua representação gráfica consiste em representar os estados ou lugares como círculos, os arcos como setas e transições como barras perpendiculares aos arcos, como mostra a Figura 10, além das marcações que são representadas como pontos dentro dos estados.

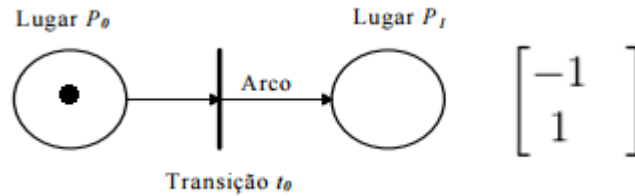


Figura 10: Rede de Petri elementar e sua matriz incidência [2]

Desta forma, a representação matricial indica que determinada transição quando disparada, removerá uma marcação de determinado estado e colocará em outro, o que pode ser dado pela equação 1.

$$M_k = M_{k-1} + A^T \cdot u_k, k = 1, 2, \quad (1)$$

Em que A é a matriz incidência correspondente a rede de Petri de dimensões $n \times m$ (sendo n o número de linhas, ou lugares, e m o número de colunas, ou transições), u_k é o vetor de disparo de m elementos, com todos os elementos igual a 0, exceto na posição da transição em que será disparada e M_k é o vetor de estados, de dimensão n , que contém quantas marcações existem em cada estado.

2.3. PNRD

A PNRD nada mais é que a representação matricial de redes de Petri elementares inseridas em uma base de dados RFID, ou seja, inserida na memória das *tags*, de maneira a permitir a identificação e atualização em tempo real do estado dos itens identificados por RFID[11].

Para se codificar a matriz incidência na *tag*, codificou-se cada elemento da matriz em 2 bits, sendo as colunas dadas como transições e as linhas dadas por estados, vetorizou-se de acordo com as linhas da matriz. Desta forma a representação é dada por:

- 00 – Se estados e transições não se relacionam (sem arcos).
- 01 – Para qual estado a transição destina.
- 10 – Para qual estado a transição se origina.

Utilizou-se desta técnica de vetorização da matriz pois a quantidade de memória disponível nas *tags* era limitado, tem-se um exemplo deste processo na Figura 11.

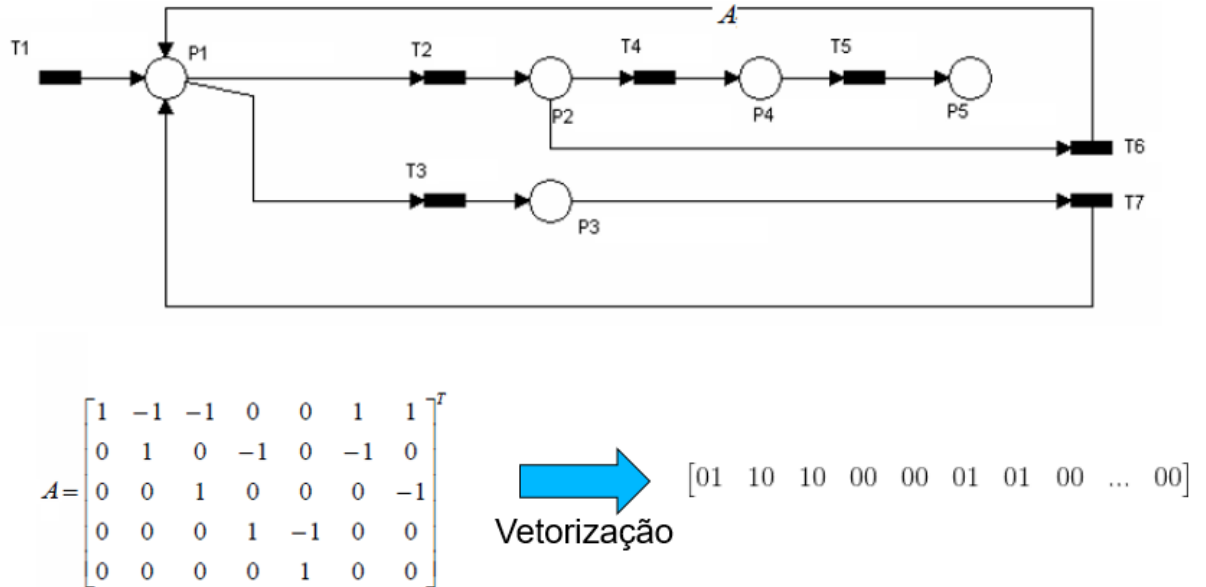


Figura 11: Exemplo de vetorização de uma matriz incidência

Além disso, a PNRD é exclusiva de cada item, ou seja, cada item pode ter um caminho diferente no processo produtivo, dependendo de qual rede de Petri está gravada na *tag*. Verificam-se também erros, ou eventos inesperados na execução da rede de Petri, de maneira a gerar ações correspondentes.

Para a PNRD se estabelece que os leitores RFID são mapeados como transições, disparando assim que encontram uma tag em um estado que a mesma possa disparar, sinalizando assim, um evento, devendo preencher algumas pré-condições vinculadas àquela transição, que quando disparada, muda o estado daquela tag de acordo com a matriz incidência gravada na tag.

Um exemplo disso pode ser dado em um sistema de entrega de mercadorias, como mostra a Figura 12.

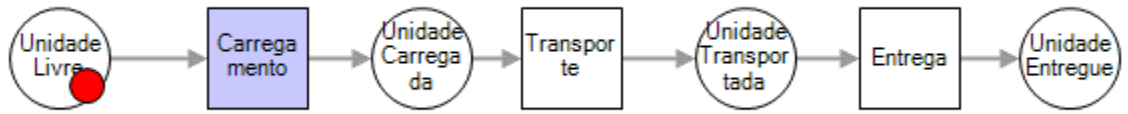


Figura 12: Representação em rede de Petri de sistema de entregas. Adaptada de [2]

A Figura 12 ilustra uma rede que, inicialmente, apresenta um *token* no estado *unidade livre*. Um leitor está vinculado à operação de *carregamento*. Se a unidade for percebida pelo leitor de carregamento, o estado do item torna-se *unidade carregada*. Caso essa unidade seja percebida pelo leitor vinculado ao caminhão, a transição *transporte* é disparada e o estado da unidade muda para *unidade transportada*. Ao chegar ao destino final, a unidade passa mais uma vez por um leitor (transição *entrega*) para, finalmente, mudar o estado para *unidade entregue*.

Esta rede de Petri, em específico, possui a seguinte matriz incidência:

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

Para exemplificar a mudança de estado *unidade livre* para *unidade carregada*, o vetor de disparos referente à transição *carregamento* é $[1,0,0]^t$. O estado inicial (*unidade livre*) é $[1,0,0,0]^t$.

O cálculo do próximo estado está representado abaixo:

$$M_1 = M_0 + A^t \cdot u_1 = [1 \ 0 \ 0 \ 0]^t + \begin{bmatrix} -1 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \cdot [1 \ 0 \ 0]^t = [0 \ 1 \ 0 \ 0]^t$$

Graficamente, a Figura 12 apresenta a mudança da marcação referente a esse disparo.

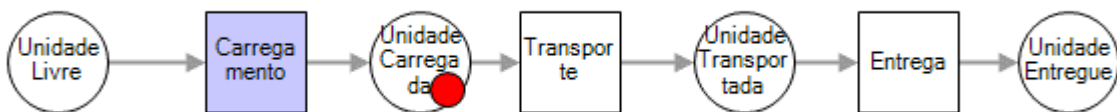


Figura 13: Nova marcação na rede de Petri de sistema de entregas. Adaptada de [2]

2.4. PNRD Estendida a Distância

A PNRD estendida a distância acrescenta o conceito de arco inibidor/ habilitador, em que são adicionadas condições de disparo a uma transição da rede de Petri. No caso da PNRD estendida a distância. Utiliza-se do cálculo da distância de [4] para determinar a mesma e utilizá-la como restrição. Nesse caso, uma transição poderá ter como pré-condição de disparo, não apenas o estado anterior habilitado, mas também uma distância ou intervalo de distância pré-determinado.

CAPÍTULO III

SISTEMA DESENVOLVIDO

3.1. Introdução

Para este projeto, foi desenvolvido o *software* DEMIS - *Distributed Environment Manufacturing Information System* ou Sistema de Informação de Manufatura em Ambiente Distribuído, e modificado o *software* RETIM – *Real Time Item Monitoring* [12]. A seguir, serão detalhados os princípios de funcionamento de ambos.

3.2. DEMIS

O *middleware* é a camada desenvolvida entre o *firmware* e o *software*, responsável por transportar informações entre diferentes plataformas, com o objetivo de abstrair a comunicação entre o *software* e o *hardware*, tornando assim, mais fácil o desenvolvimento de softwares sobre o tal. O objetivo principal deste trabalho é desenvolver um *middleware* capaz de comunicar as informações lidas pelo leitor, com qualquer mensagem pré-configurada pelo mesmo.

O DEMIS é o *middleware* servidor desenvolvido parara este trabalho que serve de camada entre o leitor RFID e algum *software* que tente extrair informações da rede (neste caso, o RETIM). O *middleware*, apesar de não ser necessário para o controle do leitor RFID, pois é possível programar em função do *firmware* do leitor diretamente, é usado em quase todos os casos para que haja uma camada de interface que facilite a comunicação entre o *software* final e o leitor RFID.

Antes de explicar o funcionamento do programa em si, deve-se definir o que é um ciclo de operação do programa, que consiste de realizar uma leitura das *tags* no ambiente, checando todas as informações gravadas nas mesmas, incluindo as matrizes incidências gravadas em cada

uma (cada matriz incidência é armazenada em uma classe Java, como mostrado no Apêndice B).

Depois, há uma verificação para checar se, além da transição atual disparar o estado da tag, algumas pré-condições são satisfeitas, para assim então, a transição poder ser disparada, escrever o novo estado (assim como foi gravado na matriz incidência da tag), atualizando o vetor de estados (dado pela classe Java no Apêndice D), e finalmente executar algumas ações, pré-parametrizadas atreladas a tal transição, este processo pode representado pelo fluxograma na Figura 14.

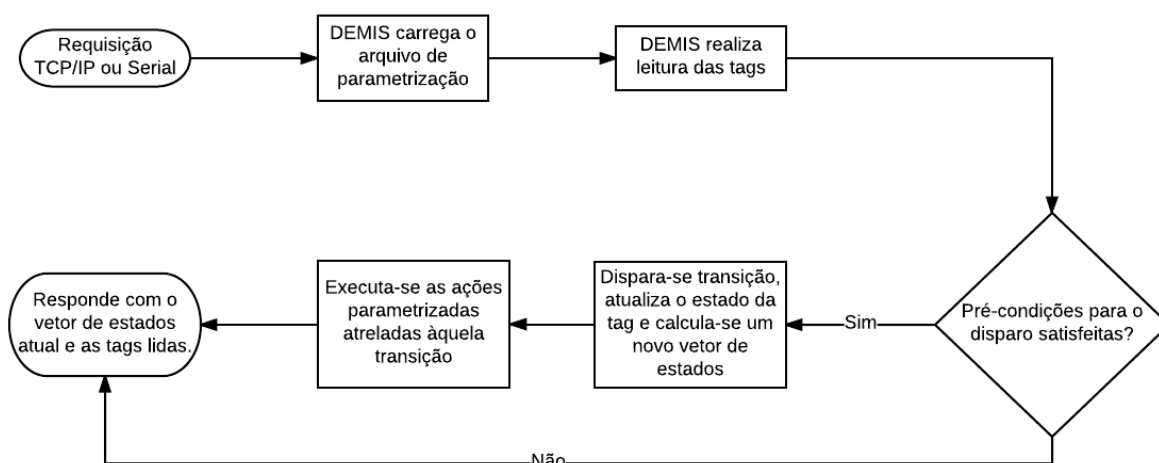


Figura 14: Fluxograma DEMIS (Fonte: Próprio Autor)

O DEMIS é um software desenvolvido em Java do tipo servidor TCP/IP (*Transmission Control Protocol* - Protocolo de Controle de Transmissão /*Internet Protocol* – Protocolo Internet), ou seja, o programa espera por conexões externas usando do protocolo TCP/IP, para assim, iniciar o ciclo de operação. Após a conexão ser estabelecida, o programa inicia um ciclo, e para cada requisição recebida, o DEMIS realiza um novo ciclo, respondendo de acordo com a parametrização pré-estabelecida.

Para que cada ciclo consiga ler as informações de forma correta, foi definida uma estrutura em que as informações seriam gravadas na *tag*, desta forma, partindo do princípio que cada leitura realizada pelo leitor lê ao menos a memória EPC, fez-se então que a matriz incidência fosse gravada no banco de memória de usuário da *tag* e que informações sobre a

matriz incidência, como o tamanho da mesma, o ID da tag e o estado atual fosse gravado no banco de memória EPC como é indicado pela Tabela 2.

Tabela 2: Estrutura de organização da memória EPC

Byte EPC	Significado
1	tagID
2	Estado Atual
3	numEstados
4	numTransições

Toda a parametrização do DEMIS foi feita em arquivos JSON (*Javascript Object Notation*) que além de permitir alta velocidade de leitura, permite um alto nível de organização e pode ser facilmente editado por qualquer um sem nenhum conhecimento prévio. A Figura 15 apresenta um exemplo de arquivo de parametrização.

O arquivo de parametrização é definido da seguinte forma: as condições necessárias para o disparo de uma transição são guardadas em um vetor “*conditions*” em que cada objeto pode conter as seguintes variáveis:

- Estado necessário (“*state*”).
- Transição a disparar (“*transition*”).
- Canal de comunicação de entrada (“*inputType*”).
- Mensagens de entrada necessárias para o disparo (“*inputMessage*”).
- Canal de comunicação de saída (“*outputType*”).
- Mensagem de saída (“*outputMessage*”).
- Distância mínima da antena (“*distanceMin*”).
- Distância máxima da antena (“*distanceMax*”).



```
{
  "conditions": [{
    "state": 0,
    "transition": 0,
    "inputMessage": ["1A Estado:1", "1B Estado:1"],
    "outputMessage": "1A1B Estado:1"
  }, {
    "state": 1,
    "transition": 1,
    "inputMessage": ["1A Estado:2", "1B Estado:2"],
    "outputMessage": "1A1B Estado:2"
  }, {
    "state": 2,
    "transition": 2
  }],
  "inputType": "tcpip",
  "outputType": "serial",
  "host": "192.168.0.120",
  "port": 9001
}
```

Figura 15: Exemplo de arquivo de parametrização (Fonte: Próprio autor)

As distâncias são calculadas baseadas em um trabalho prévio[5] em que a distância é calculada baseada no RSSI (potência do sinal recebido) da tag e são parametrizadas em centímetros.

Além disso, pode-se definir o tipo de comunicação que será usado para entrada e saída; para isto, basta definir os parâmetros “*inputType*” e “*outputType*” respectivamente, bem como o *IP* e porta para o qual será transmitido a mensagem de saída, no caso de saída via TCP/IP ou o número da porta serial indexada pelo sistema operacional (desta maneira, as funcionalidades serial operam por enquanto apenas em sistema operacional Windows), no caso de comunicação via serial.

Caso não seja definida nenhuma mensagem para ser enviada, ou seja, nenhuma transição seja disparada, uma mensagem padrão será enviada caso esta esteja definida no arquivo de parametrização fora do vetor “*conditions*”, do contrário, é enviada a mensagem padrão contendo apenas vetor de estados e as *tags* lidas no ambiente.

3.3. RETIM

O RETIM é um software para visualização em tempo real do estado e processo que um item identificado por RFID contendo a estrutura da PNRD interna. O RETIM previamente desenvolvido em Java, utilizando como base a biblioteca JGraphX, capaz de exibir grafos e diagramas [12]. A partir de uma sequência de *bytes* simbolizando uma matriz incidência, o mesmo apresenta uma representação gráfica correspondente a esta matriz na tela. Sendo que, as transições são dadas representadas como quadrados azuis, os estados como círculos verdes e os arcos como setas. Além disso o programa mostra a marcação da *tag* (indicando qual estado a *tag* se encontra) como um círculo preto ao centro do círculo do estado. Nota-se que deste modo, é possível representar apenas uma *tag* por vez nesta rede de Petri, uma vez que não há nenhuma representação que possa indicar o número de marcações por estado do modo em que o programa foi construído. A Figura 16 apresenta uma representação gráfica de dados coletados de uma *tag* com 4 estados e quatro transições, tendo o estado atual em *P4*.

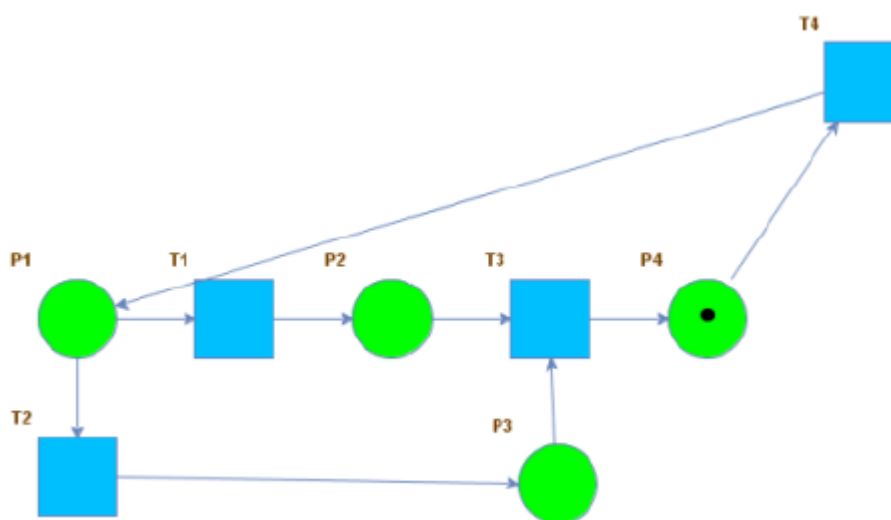


Figura 16: Representação gráfica pré-modificações RETIM (Fonte: Próprio autor)

Foram feitas então algumas modificações no código inicial para que fosse possível a comunicação com o DEMIS via TCP/IP, além de adicionar uma indicação com o número de marcações para cada estado (o símbolo de marcação antigo continua para representar o estado da primeira *tag* lida) como mostrado na Figura 17, e uma pequena interface, ao clicar sobre determinado estado, que mostra informações sobre as *tags* naquele estado (atualmente, somente o que está gravado na memória EPC da *tag* em formato hexadecimal), como se pode ver na Figura 18.

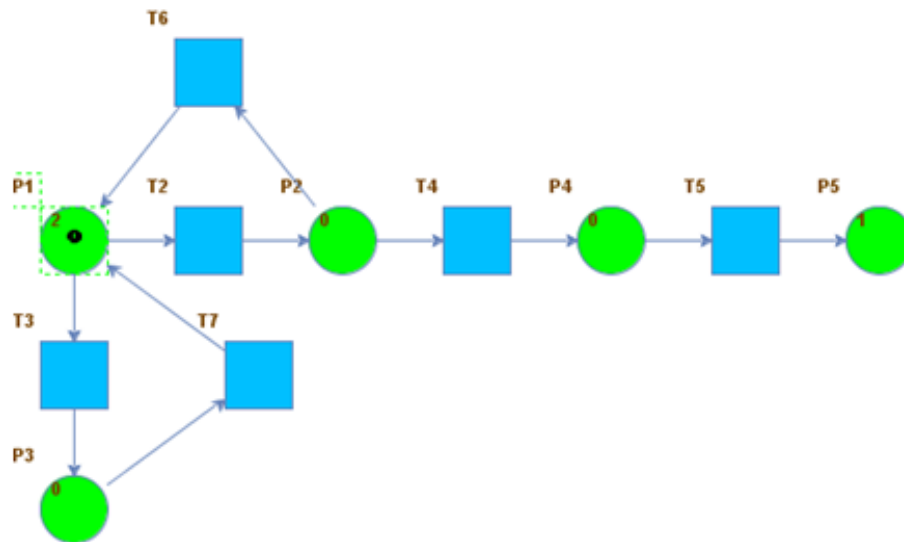


Figura 17: Representação gráfica pós-modificações RETIM (Fonte: Próprio autor)

INFO			
n	EPC		
1	AB000507		
2	00000507		

Figura 18: Informações das tags no estado 00 (Fonte: Próprio autor)

Como está representado na Tabela 2 e na Figura 18, ao clicar-se no estado P1, são mostradas as informações das *tags* naquele estado. Neste caso, existem duas tags, sendo a primeira tag com *id* “AB” e a segunda com *id* “00”, ambas no estado inicial “00”, e com sua informação quanto ao número de estados e de transições representadas logo em seguida, neste caso, cinco estados e sete transições.

CAPÍTULO IV

RESULTADOS

4.1. Introdução

Toda a comunicação com o leitor foi feita com base na Mercury API em Java, desenvolvida pela Thingmagic (empresa do leitor RFID). Usando da mesma, é possível executar operações de leitura e escrita em tags por meio de qualquer leitor RFID da Thingmagic conectado a uma antena RF.

Para a leitura das *tags* foi usada uma estrutura montada usando o leitor m6e da *thingmagic* já fornecida pelo laboratório MAPL, com interfaceamento USB com o computador, comunicando com as *tags* através de uma antena UHF, como ilustra a Figura 19.

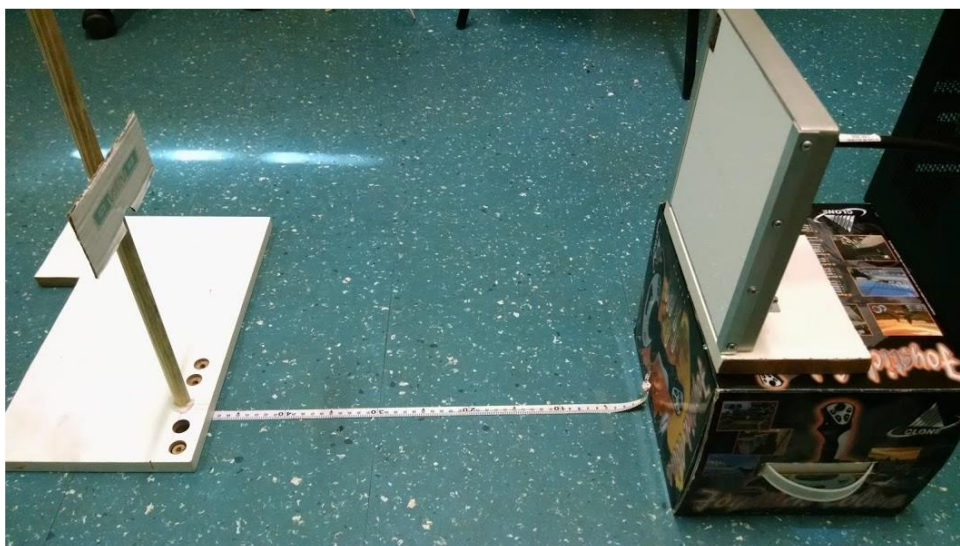


Figura 19: Estrutura de testes para PNRD

As *tags* utilizadas (Figura 20), são *tags* passivas UHF tradicionais (microchip, antena e invólucro) como é mostrada, e possuem a sua memória dividida em quatro bancos organizada conforme apresentado na Figura 4.



Figura 20: Tag UHF utilizada

Para gravar os dados da PNRD na tag, foram usados dos bancos de memória EPC e de usuário que a tag oferecia. Para a tag utilizada, não havia um *datasheet* com as informações referentes a mesma, então por meio de testes, verificou-se que a tag utilizada, possui 12 bytes para o banco EPC (exceto os bytes de controle).

Como a API utilizada, possui palavra de 16 bits, ou seja, só era possível ler e escrever na *tag* 16 bits por vez, criou-se uma classe Java (Apêndice C) para facilitar a modificação de apenas alguns bits da tag, assim que houvesse uma leitura, que é representada por outra classe Java (Apêndice E).

A convenção, para este tipo de tag, é de se usar este banco para identificação de produtos, seguindo a norma “EPC Global Gen 2”. Porém, como este banco é caracterizado por ser de leitura e escrita, este trabalho se utilizou de 4 bytes deste banco, a fim de melhorar a organização dos dados. Fazendo com que a tag armazenasse dados sobre a rede de Petri que estava inserida, como mostrado na Tabela 2.

O banco de Usuário, quando existe, geralmente é o maior banco da tag e, da mesma maneira que foi feito com o banco EPC, por meio de testes, descobriu-se que este banco possui 64 bytes de armazenamento. E é neste banco que é guardada a estrutura da rede de Petri, propriamente dita

O leitor utilizado tem a capacidade de realizar leituras e escritas na memória das *tags* de acordo com mensagens enviadas para o mesmo. Este então interpreta a mensagem, a processa e então executa uma ação correspondente, podendo realizar até 750 operações (leitura ou escrita) por segundo, com uma potência de saída de até 30dBm, que com uma antena de 6dbi pode significar em uma distância de leitura de até 9 metros[13].

A antena utilizada possui 9dbi, sendo assim, a distância de operação era mais do que o suficiente para se realizar os testes quanto à lógica da PNRD.

4.2. Casos de uso

Para testar o sistema desenvolvido, foi usado um caso de uso conforme Figura 21.

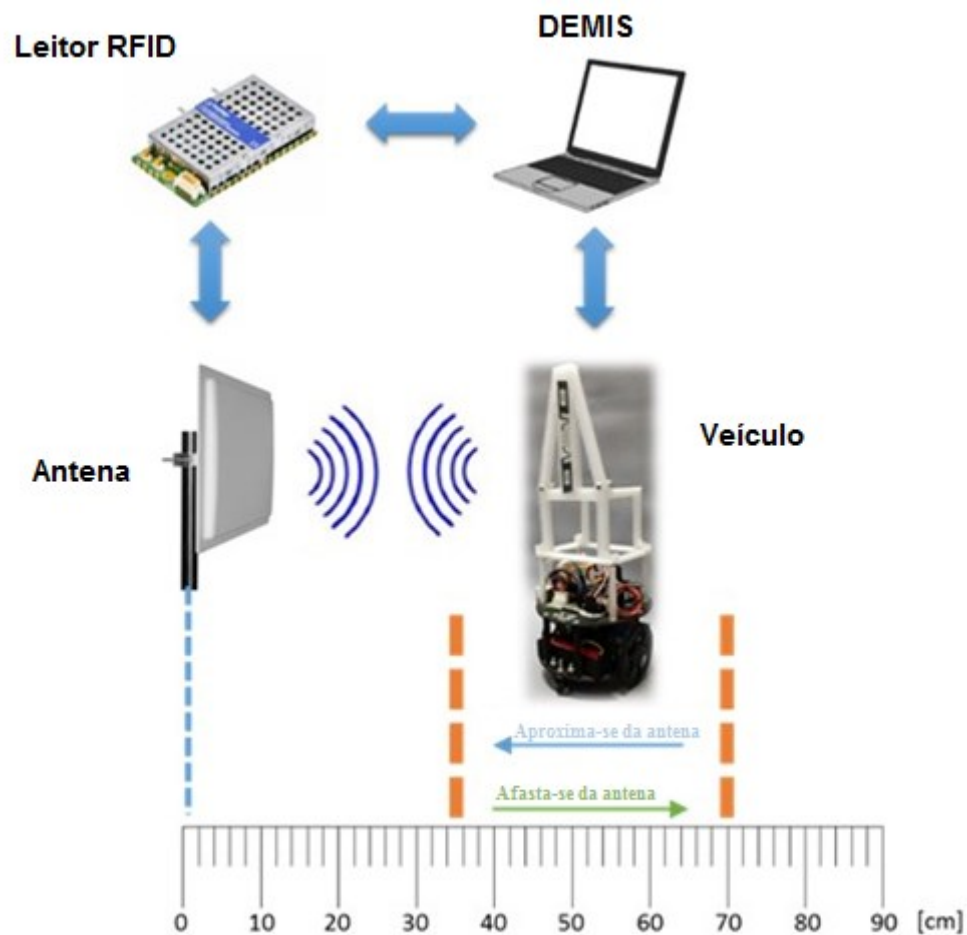
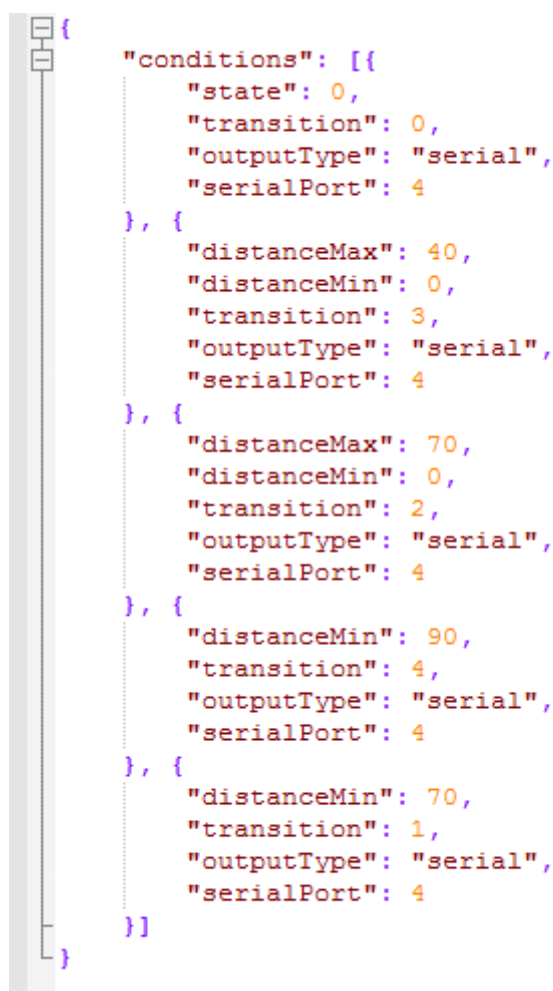


Figura 21: Montagem do caso de uso. Adaptado de [14]

Como ilustra a Figura 14, inicialmente inicia-se o servidor DEMIS. O servidor então fica esperando até que uma conexão seja realizada ao mesmo. Depois de estabelecida a conexão,

e definida através do arquivo de configuração. É iniciada a leitura das *tags* na área de atuação da antena, e assim que alguma tag satisfaça alguma das condições pré-estabelecidas, tal transição é disparada.

Usando de um robô móvel equipado de um arduino, conectado ao computador via USB, com um suporte para a *tag*, onde o DEMIS é executado, por sua vez, conectado ao leitor RFID que comanda a antena. Estabeleceu-se que o veículo deveria realizar um movimento cíclico de se aproximar e afastar da antena. Desta forma, a condição para o veículo se aproximar, seria a distância ser maior ou igual a 70cm da antena, e conseqüentemente, a condição para o veículo se afastar seria a distância menor que 70cm da antena. Tem-se o seguinte arquivo de configuração (Figura 22).



```
{
  "conditions": [{
    "state": 0,
    "transition": 0,
    "outputType": "serial",
    "serialPort": 4
  }, {
    "distanceMax": 40,
    "distanceMin": 0,
    "transition": 3,
    "outputType": "serial",
    "serialPort": 4
  }, {
    "distanceMax": 70,
    "distanceMin": 0,
    "transition": 2,
    "outputType": "serial",
    "serialPort": 4
  }, {
    "distanceMin": 90,
    "transition": 4,
    "outputType": "serial",
    "serialPort": 4
  }, {
    "distanceMin": 70,
    "transition": 1,
    "outputType": "serial",
    "serialPort": 4
  }
  ]
}
```

Figura 22: Arquivo de configuração usado. Adaptado de [14]

A PNRD estendida correspondente ao robô móvel pode ser vista na Figura 23. O veículo foi programado para se aproximar da antena quando a marcação estivesse no estado P3 e se afastar quando a marcação estivesse no estado P4. Desta forma, quando a *tag* contida no veículo iniciaria no estado de inicialização P1, transacionaria imediatamente para o estado P2 em que seria tomada a decisão de qual transição deveria ser disparada em função da distância entre a *tag* e a antena. Escolhido o estado, o veículo então se movimentaria até atingir certo limite (neste caso 40cm ou 90cm), para então retornar ao estado de decisão P2.

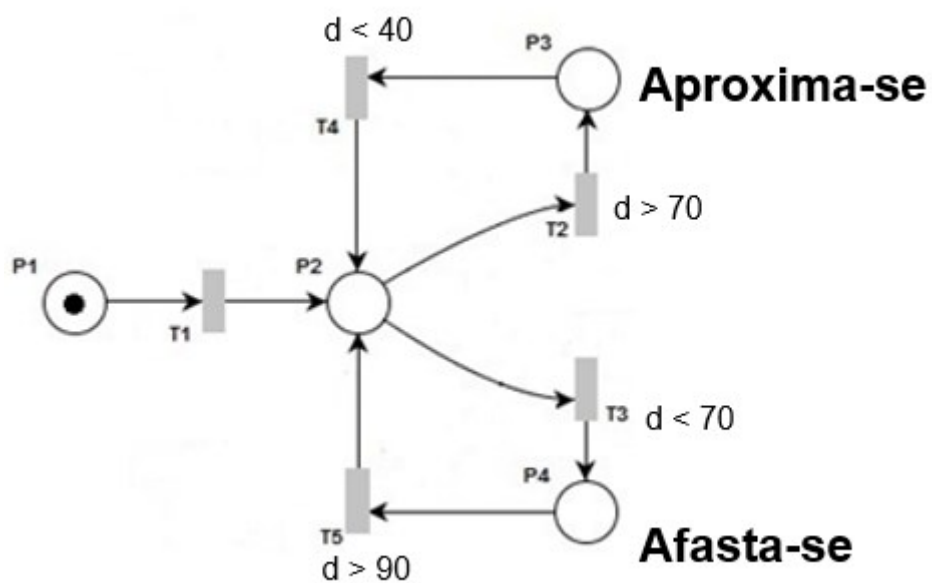


Figura 23: PNRD estendida a distância referente ao controle do Robô Móvel. [14]

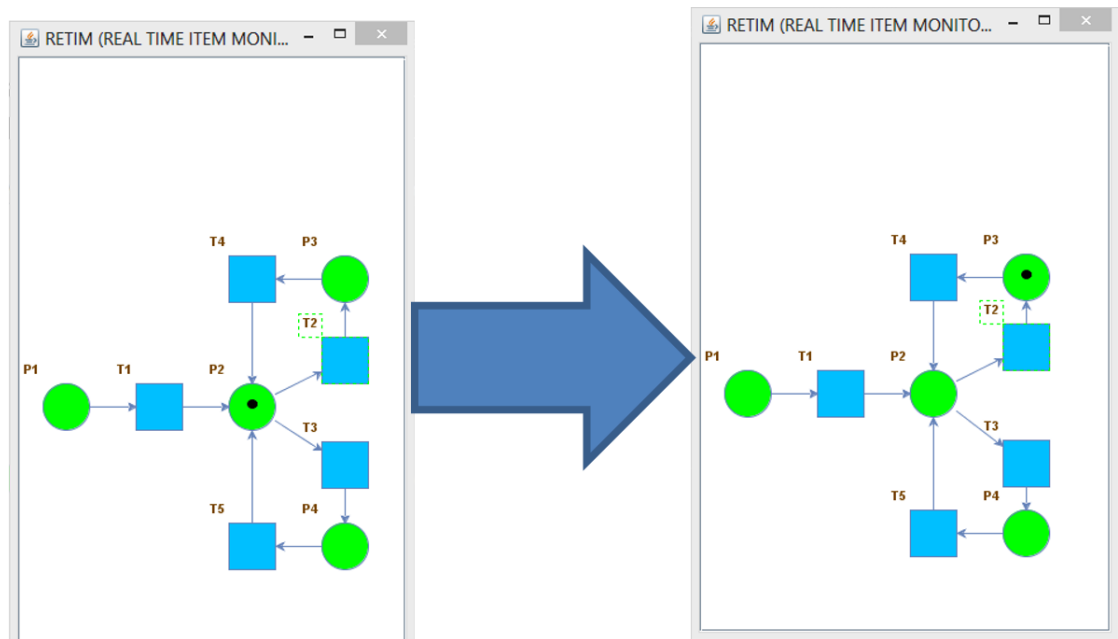


Figura 24: Exemplo de Transição T2 sendo disparada, depois de decisão. [14]

A Figura 24 ilustra o modelo representado pela Figura 23 em que a transição T2 é disparada, representado pelo RETIM Com um arquivo de configuração, determina-se todas as condições necessárias para que uma transição seja disparada, desta forma, desde que todos os requerimentos sejam satisfeitos, a transição é disparada, trocando o estado da tag,

Depois de disparada a transição, é enviada uma mensagem, também pré-configurada, correspondente àquela transição, além de que um evento para que atualize a tela do RETIM de maneira correspondente, neste caso, não foi definida nenhuma mensagem para ser enviada, então, a mensagem enviada é o número inteiro indicando o estado atual da tag.

Inicialmente a *tag* começaria no estado P1, que logo transicionaria para o estado P2. A marcação poderia então ir ou para o estado P3, ou P4 em função da distância da tag até a antena. No caso da Figura 24 é ilustrada a *tag* transicionando do estado de decisão P2, para o estado de aproximação P3, disparado pela transição T2 (requer distância maior que 70cm), depois disso, o veículo se aproxima da antena até que seja atingida uma distância de ao menos 40cm, disparando assim, a transição T4, retornando ao estado de decisão, que desta vez, devido a distância atual ser de 40cm (distância menor que 70cm), seria então disparada a transição T3, e o veículo começaria a se afastar até que estivesse a pelo menos 90cm da antena, disparando então a transição T5, e retornando para o estado de decisão P2 e reiniciando o ciclo conforme foi definido pelo modelo da Figura 23.

CAPÍTULO V

CONCLUSÕES

Desta forma, conclui-se então que o objetivo final deste trabalho foi atingido (com exceção do desenvolvimento do algoritmo de PNRD e PNRD estendida a distância), pois foi possível implementar e validar todas as funcionalidades necessárias para que o PNRD funcionasse, ou seja, persistência dos dados nas *tags*, disparo das transições correspondendo corretamente ao arquivo de configuração,

Demonstra-se também que a tecnologia RFID apresenta muito potencial para, além de permitir um banco de dados de forma distribuída, permite que todo o controle de uma cadeia de produção seja feito baseado nas informações armazenadas. O que encaixa perfeitamente com o conceito de “internet das coisas”, ou seja, um sistema completamente descentralizado.

Para continuar esta linha de pesquisa, para trabalhos futuros, seria complementado o algoritmo afim de que o mesmo, também seria capaz de interpretar redes de PNRD estendida a montagem e desmontagem, ou seja, seria adicionado suporte para processos que incluem montagem e desmontagem de peças.

REFERÊNCIAS BIBLIOGRÁFICAS

- [1] Tavares, J. J. P. Z. S.; Saraiva, T. A. (2010) ELEMENTARY PETRI NET INSIDE RFID DISTRIBUTED DATABASE (PNRD). International Journal Of Production Research, V.48, N.9, P.25632582, 2010
- [2] Azambuja, Marcelo Cunha De (2011). Modelos E Técnicas Para Simulação De Sistemas Uhf De Identificação Por Rádio Frequência (RFID)/ Marcelo Cunha De Azambuja. – Tese De Doutorado – Porto Alegre, 2011, 136 P.
- [3] Ngai, E.W.T., Moon, K.K.L., Riggins, F.J., Yi, C.Y. RFID research: An academic literature review (1995-2005) and future research directions.
- [4] MUROFUSHI, R. H. (2016), Developing of an indoor positioning system based on the received signal strength indication of a RFID system. 111 f. M. Sc. Dissertation, Universidade Federal de Uberlândia, Uberlândia.
- [5] <http://blog.atlasrfidstore.com>, acesso em 26/09/2016
- [6] Saab S. S., Nakad Z. S. (2011) A STANDALONE RFID INDOOR POSITIONING SYSTEM USING PASSIVE TAGS. Ieee Transactions On Industrial Electronics, Vol. 58, No. 5
- [7] HESSEL, Fabiano; DIAS, Renata Rampim de Freitas ; VILLAR, Reinaldo Serrano Goy. Implementando RFID na Cadeia de Negócios. 2. ed. [S.l.]: Edipucrs, 2011. 344 p.
- [8] <http://www.rfidsb.com/secrets-of-the-select-command>, acesso em 07/05/2016.
- [9] http://coral.ufsm.br/gpscom/professores/andrei/Teoria/aula_06new.pdf, acesso em 20/02/2017
- [10] http://www.dca.ufrn.br/~affonso/DCA0409/pdf/redes_de_petri.pdf, acesso em 11/02/2017
- [11] Silva G. P., Tavares J. J. P. Z. S. (2012) Sistema De Montagem E Desmontagem Baseado Em Pnrd
- [12] Molina F. S., Marcos W. P., Sousa A. R., Tavares J. J. P. Z. S. (2012) Algoritmo Para Visualização De Processos De Manufatura Baseados Na Pnrd. Anais Do Xix Congresso Brasileiro De Automática
- [13] <http://www.thingmagic.com/images/pdfs/micro-datasheet.pdf> acesso em 15/02/2017
- [14] Tavares, J.J.P.Z.S., Murofushi, R.H., Silva, L.H., Silva, G. R. Petri Net Inside RFID Database Integrated with RFID Indoor Positioning System for Mobile Robots Position

Control. In Proceedings of PNSE 2017, Zaragoza, 2017. Disponível em <http://ceur-ws.org/Vol-1846/paper10.pdf>. Acessado em 02/10/2017.

APÊNDICE A – DEMIS CORE

```
package continuous_adm;

import Extra.EncodeDecode;
import Extra.JavaCPN;
import Extra.TagRead;
import com.fazecast.jSerialComm.SerialPort;
import com.thingmagic.*;
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.ObjectOutputStream;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.ConnectException;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.logging.Level;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;
```

```

/**
 *
 * @author Lucas Programa que checa a matriz incidência da Tag, procura por
 * relação e, caso sim, grava novo estado.
 * read()->readInternal()->MessageEmbedded - MsgAddGEN2(EDITA MSG, com INFO do
 * plano) -> SendTimeout()
 */
public class Monitor_Tags {

    static String URI = "tmr:///com7";
    static int RETIM_PORT = 7070;
    static int HF_PORT = 8080;
    static int CPN_PORT = 9001;
    static int[] antennaList = new int[]{2};
    static int transition = 3;
    static Reader r;
    static Vector statesVector;
    static ObjectOutputStream objOut = null;
    static JSONParser parser;
    static JSONObject pnr;
    static JSONArray conditions;
    static JSONArray coefficients;
    static Socket s = null;
    static Socket globalSocket = null;
    static ServerSocket HFServer = null;
    static BufferedOutputStream bos = null;
    static JavaCPN cpn = new JavaCPN();
    static ArrayList inputCollection = new ArrayList();

    /**

```

```

* Método que inicia a leitura toda vez que chega uma requisição(byte igual
* a 42). Encerra as atividades, caso receba um byte menor que 0. Lê
* continuamente o id, o estado atual, o numero de estados e transições,
* armazenados em ordem nos primeiros bytes da memória EPC da tag. Além de
* armazenar a matriz incidência na memória da de usuário da tag.
*
* @param server Parâmetro de servidor que fica esperando até que chegue uma
* requisição
* @throws IOException
* @throws ReaderException
* @throws ParseException
*/

public static void readTags(ServerSocket server) throws IOException, ReaderException,
ParseException {
    byte lenght;
    IncidentMatrix incident = null;
    TagReadData readData[];
    byte[] b = new byte[300];
    int states = 0, transitions = 0, check;
    Socket s = server.accept();
    short i;
    while (true) {
        if ((check = s.getInputStream().read()) >= 128) {           //DESLIGAR O DEMIS
            System.out.println("DESLIGANDO DEMIS");
            break;
        } else if (check == 42 && states > 0 && transitions > 0 && incident != null && r !=
null && s.isConnected() == true) {
            refreshTags(states, transitions, incident, r, s);
            continue;
        }
        objOut = new ObjectOutputStream(s.getOutputStream());
    }
}

```

```

objOut.flush();

Gen2.ReadData readEpcOp = new Gen2.ReadData(Gen2.Bank.EPC, 2, (byte) 2);

short[] entry = (short[]) r.executeTagOp(readEpcOp, null);

b[0] = (byte) ((entry[1] & 0xFF00) >> 8);

states = b[0];                                     //NUM ESTADOS

System.out.println("ESTADOS " + b[0]);

b[1] = (byte) (entry[1] & 0x00FF);

transitions = b[1];                                //NUM TRANSICOES

System.out.println("TRANSICOES " + b[1]);

System.out.println("ESTADO-1ª " + (entry[0] & 0x00FF));    //ESTADO DA 1ª
TAG

System.out.println("TRANSICAO: " + transition);

s.getOutputStream().write(b, 0, 2);                //ESCREVE ESTADOS E
TRANSICOES

s.getOutputStream().flush();

statesVector = new Vector(states);

length = (byte) Math.ceil(((double) (states * transitions) / 8);    //NÚMERO DE
PALAVRAS

Gen2.ReadData readDataOp = new Gen2.ReadData(Gen2.Bank.USER, 0, length);

short[] matrix;

SimpleReadPlan plan = new SimpleReadPlan(antennaList, TagProtocol.GEN2, null,
readDataOp, 1000);

r.paramSet(TMConstants.TMR_PARAM_READ_PLAN, plan);

matrix = (short[]) r.executeTagOp(readDataOp, null);

incident = new IncidentMatrix(matrix, states, transitions);    //PUXA MATRIZ
INCIDENCIA - 1º TAG

s.getOutputStream().write(incident.getBytes());    //ESCREVE A MATRIX
INCIDÊNCIA

s.getOutputStream().flush();

readData = r.read(100);

for (TagReadData v : readData) {                    //EM TODAS AS TAGS

    int relation;

```

```

//System.out.println(ReaderUtil.byteArrayToHexString(v.getData()));

byte state = v.getTag().epcBytes()[1]; //PUXA ESTADO ATUAL
DA TAG - LSB da Palavra 1 EPC

statesVector.setValue(state, statesVector.getValue(state) + 1);

System.out.println("STATE: " + state + "  TRANSITION: " + transition + "\n");

if ((relation = incident.getValue(state, transition)) == 2) { //SE HOUVER
RELAÇÃO TR-ST

    System.out.println("HÁ RELAÇÃO ESTADO-TRANSIÇÃO");

    Vector uK = new Vector(incident, transition); //VETOR COLUNA

    System.out.print("VETOR DE ESTADOS: ");

    statesVector.printVector();

    System.out.print("VETOR COLUNA(DISPARIOS: ");

    uK.printVector();

    for (i = 0; i < uK.statesQuantity.length; i++) { //CHECA NOVO
ESTADO

        if (uK.getValue(i) == 1) {

            break;

        }

    }

    statesVector.sumVector(uK); //NOVO VETOR DE
ESTADOS - TODO:SETAR NOVO VETOR DE ESTADOS APENAS SE GRAVOU
NOVO ESTADO

    //*****//A
API NÃO SINALIZA, DEVE-SE LER DENV PRA CHECAR...

    short newState[] = new short[] {i}; //NOVO ESTADO

    Gen2.WriteData writeEpcOp = new Gen2.WriteData(Gen2.Bank.EPC, 2,
newState);

    System.out.println("NOVO ESTADO: " + newState[0]);

    System.out.print("NOVO VETOR DE ESTADOS: ");

    statesVector.printVector();

    //r.executeTagOp(writeEpcOp, v.getTag()); //GRAVA NOVO
ESTADO NAS TAGs APLICAWEIS TODO: SÓ ATUALIZAR VETOR DE ESTADOS,
CASO GRAVACAO DE TAG SUCCESSFULL

```

```

    } else if (relation < 0) {
        System.out.println();
    } else {
        System.out.println("SEM RELAÇÃO ESTADO-TRANSIÇÃO: " + relation);
    }
}

byte[] numTags = new byte[statesVector.getStatesQuantity().length]; //CRIANDO
VETOR DE ESTADOS EM BYTES.

for (i = 0; i < numTags.length; i++) {
    numTags[i] = (byte) statesVector.getStatesQuantity()[i];
}

System.arraycopy(numTags, 0, b, 0, states); //COPIA O VETOR DE
ESTADOS

s.getOutputStream().write(b, 0, states); //ESCREVE O VETOR DE
ESTADOS

s.getOutputStream().flush();
}
s.close();
}

/**
 * Obtem a transição correspondente para determinada Tags
 *
 * @param tag Leitura de uma tag
 * @return transição para determinadas condições, retorna -1, caso não
 *         encontre nenhuma transição válida.
 * @throws java.io.IOException
 * @throws org.json.simple.parser.ParseException
 */
public static int getTagTransition(TagRead tag) throws IOException, ParseException {
    int state = tag.getState();

```

```

int transition = -1;

int tagID = tag.getID(); //PUXA TAGID DA TAG

int rssi = tag.getRssi();

int frequency = tag.getFrequency();

int lightSpeed = 299792458; //velocidade da luz em m/s

double phase = 1 - (double) tag.getPhase() / 180;

double lambda = ((lightSpeed) / (double) (4 * frequency) / 10.0);
//lambda em cm

Double distanceRssi = new Double(-1);

for (Object c : coeficients) {
    JSONObject cft = (JSONObject) c;

    Long jsonFrequency = (long) cft.get("FREQUENCY");

    if (frequency == jsonFrequency) {
        double c0 = (double) (cft.get("c0"));
        double c1 = (double) (cft.get("c1"));
        double c2 = (double) (cft.get("c2"));

        distanceRssi = ((-c1 - Math.sqrt(Math.pow(c1, 2) - 4 * c2 * (c0 - rssi))) / (2 * c2));

        break;
    }
}

if (distanceRssi < 0 || distanceRssi.isNaN()) {
    System.out.println("ERRO AO MEDIR A DISTÂNCIA: DISTÂNCIA MENOR QUE
0 ou NaN: " + distanceRssi);

    return -1;
}

double numOnda = Math.floor(distanceRssi / lambda);

double distance = (numOnda + phase) * lambda;

System.out.println("Distância RSSI: " + distanceRssi);

int antenna = tag.getAntenna();

String inputMessage = "";

if (HFServer == null || HFServer.isClosed()) {

```

```

        HFServer = new ServerSocket(HF_PORT);

        s = HFServer.accept();

    }

    String jsonInputType = pnr.get("inputType") == null ? "tcpip" : (String)
    pnr.get("inputType");

    if (jsonInputType.equals("tcpip")) {

        BufferedReader bis = new BufferedReader(new
        InputStreamReader(s.getInputStream()));

        inputMessage = bis.readLine();

        inputMessage = inputMessage.trim();

        System.out.println("INPUT: " + inputMessage);

//        HFServer.close();

    }

    for (Object c : conditions) {

        JSONObject cdt = (JSONObject) c;

        Long jsonId = (Long) cdt.get("id");           //PUXA TAGID DO JSON

        Long jsonDistanceMin = (Long) cdt.get("distanceMin");

        Long jsonDistanceMax = (Long) cdt.get("distanceMax");

        Long jsonAntenna = (Long) cdt.get("antenna");

        Long jsonState = (Long) cdt.get("state");

        JSONArray jsonInputMessage;

        try{

            jsonInputMessage = (JSONArray) cdt.get("inputMessage");

        } catch (NullPointerException e){

            jsonInputMessage = new JSONArray();

        }

        if ((jsonId == null || tagID == jsonId)

            && (jsonState == null || state == jsonState)

            && (jsonDistanceMin == null || distance >= jsonDistanceMin)

            && (jsonDistanceMax == null || distance <= jsonDistanceMax)

            && (jsonAntenna == null || antenna == jsonAntenna)) {

```

```

        inputCollection.add(inputMessage);
        if (inputCollection.containsAll(jsonInputMessage)) {
            transition = ((Long) (cdt.get("transition"))).intValue(); //E SUA
TRANSIÇÃO CORRESPONDENTE ÀS CONDIÇÕES
//        PrintWriter pw = new PrintWriter(s.getOutputStream());
//        pw.write((String)cdt.get("outputMessage"));
//        pw.flush();
            JavaCPN jCPN = new JavaCPN(s);
            jCPN.send(EncodeDecode.encode((String) cdt.get("outputMessage")));
            inputCollection.clear();
            break;
        }
        else
            try{
                if(!jsonInputMessage.contains(inputCollection.get(inputCollection.size() - 1)))
                    inputCollection.remove(inputCollection.size()-1);
            } catch(IndexOutOfBoundsException e){
            }
        }
    }
    return transition;
}

/**
 * Executa uma pós condição, neste caso, enviar uma mensagem. Chamar este
 * método apenas quando já foi determinado qual transição deverá ser
 * chamada.
 *
 * @param state estado que será transmitido
 * @param transition transição atual do programa

```

```

* @throws FileNotFoundException
* @throws IOException
* @throws ParseException
*/

public static void executePosCondition(int state, int transition) throws
FileNotFoundException, IOException, ParseException {
    for (Object c : conditions) {
        JSONObject condition = (JSONObject) c;
        if (((Long) condition.get("transition")).intValue() == transition) {
            String outputType = condition.get("outputType") == null ? "" : (String)
condition.get("outputType");
            if (outputType.equals("serial")) {
                Long port = (Long) condition.get("serialPort");
                SerialPort comPort = SerialPort.getCommPort("COM" + port); //APENAS
WINDOWS
                if (!comPort.isOpen()) {
                    while (!comPort.openPort()) {
                        System.out.println("NAO FOI POSSÍVEL ABRIR A PORTA COM" +
port);
                    }
                }

                comPort.setComPortTimeouts(SerialPort.TIMEOUT_READ_SEMI_BLOCKING, 100, 0);
                System.out.println("PORTA SERIAL COM" + port + " ABERTA COM
SUCESSO");
            }
            OutputStream out = comPort.getOutputStream();
            out.write(state);
            out.flush();
        } else if (outputType.equals("tcpip")) {
            String host = (String) condition.get("host");
            Long port = (Long) condition.get("port");
            try {

```

```

        if (condition.get("outputType") == null) {
            s = globalSocket;
        } else if (host != null && port != null) {
            s = new Socket(host, port.intValue());
        } else {
            System.out.println("ERRO: HOST/PORTA NAO PREENCHIDOS");
            return;
        }

        BufferedOutputStream buf = new
BufferedOutputStream(s.getOutputStream());

        buf.write(state);
        buf.flush();
    } catch (UnknownHostException e) {
        System.out.println("HOST NÃO ENCONTRADO");
    } catch (ConnectException e) {
        System.out.println("HOST NÃO ALCANÇADO");
    }
} else if (pnrd.get("outputType").equals("cpn")) {
    try {
        cpn.send(EncodeDecode.encode((String) condition.get("outputMessage")));
        cpn.send(EncodeDecode.encode("1"));
    } catch (NullPointerException e) {
        System.out.println("MENSAGEM CPN NÃO ENVIADA COM SUCESSO");
    }
}
break;
}
}
}

```

```

/**
 * Método que realiza uma leitura de tags no ambiente e retorna qual o
 * estado que o sistema se encontra. Escreve o novo vetor de estados e todas
 * as informações de todas as tags no socket passado, além de realizar a
 * mudança de estados.
 *
 * @param states número de estados da matriz incidência
 * @param transitions número de transições da matriz incidência
 * @param incident Objeto que referencia a matriz incidência
 * @param r Objeto que referencia o leitor
 * @param s Socket para receber as requisições
 * @throws IOException
 * @throws ReaderException
 * @throws ParseException
 */

public static void refreshTags(int states, int transitions, IncidentMatrix incident, Reader r,
Socket s) throws IOException, ReaderException, ParseException {
    statesVector.reset();

    short i;                                //NOVO ESTADO
    int j = 0;
    byte[] b = new byte[15];
    TagReadData[] readData = r.read(100);    //LÊ POR 100ms
    TagRead[] tagReads = new TagRead[readData.length];
    for (TagReadData v : readData) {         //PARA CADA TAG
        int relation;
        tagReads[j] = new TagRead(v);
        int state = tagReads[j].getState();   //PUXA ESTADO ATUAL DA
TAG - LSB da Palavra 1 EPC
        //System.out.println(ReaderUtil.byteArrayToHexString(v.getData()));
        transition = getTagTransition(tagReads[j]);
        if (transition < 0) {

```

```

        System.out.println("ERRO AO SE OBTER A TRANSIÇÃO ATUAL DA TAG: " +
v.epcString());
    } else {
        executePosCondition(state, transition);
        statesVector.setValue(state, statesVector.getValue(state) + 1);
        System.out.println("STATE: " + state + "  TRANSITION: " + transition + "    ID:
" + tagReads[j].getID() + "\n");
        incident.printMatrix();
        System.out.println();
        if ((relation = incident.getValue(state, transition)) == 2) { //SE HOUVER
RELAÇÃO TR-ST
            System.out.println("HÁ RELAÇÃO ESTADO-TRANSIÇÃO");
            Vector uK = new Vector(incident, transition);           //VETOR COLUNA
            System.out.print("VETOR DE ESTADOS: ");
            statesVector.printVector();
            System.out.print("VETOR COLUNA: ");
            uK.printVector();
            for (i = 0; i < uK.statesQuantity.length; i++) {        //CHECA NOVO ESTADO
                if (uK.getValue(i) == 1) {
                    break;
                }
            }
            statesVector.sumVector(uK);                               //NOVO VETOR DE
ESTADOS - TODO:SETAR NOVO VETOR DE ESTADOS APENAS SE GRAVOU
NOVO ESTADO
            //*****//A
API NÃO SINALIZA, DEVE-SE LER DENV PRA CHECAR...                //NOVO
ESTADO

            Gen2.WriteData writeEpcOp = new Gen2.WriteData(Gen2.Bank.EPC, 2, new
short[] {i}); // i = novoEstado
            System.out.println("NOVO ESTADO: " + i);
            System.out.print("NOVO VETOR DE ESTADOS: ");
            statesVector.printVector();

```

```

        r.executeTagOp(writeEpcOp, v.getTag());           //GRAVA NOVO
ESTADO NAS TAGs APLICAVEIS

        tagReads[j].setState(i);

    } else if (relation < 0 || relation > 2) {

        System.out.println("MATRIZ INCIDÊNCIA MAL FORMADA: ");

        incident.printMatrix();

    } else {

        System.out.println("SEM RELAÇÃO ESTADO-TRANSIÇÃO(Nenhuma
Transição Disparada): " + relation);

    }

    j++;

}

byte[] numTags = new byte[statesVector.getStatesQuantity().length]; //CRIANDO
VETOR DE ESTADOS EM BYTES.

for (i = 0; i < numTags.length; i++) {

    numTags[i] = (byte) statesVector.getStatesQuantity()[i];

}

System.arraycopy(numTags, 0, b, 0, states);           //COPIA O VETOR DE
ESTADOS

s.getOutputStream().write(b, 0, states);               //ESCREVE O VETOR DE
ESTADOS - TODO: ESCREVER UM OBJETO DA LEITURA

s.getOutputStream().flush();

objOut.writeObject(tagReads);

objOut.flush();

}

public static void main(String[] args) throws IOException, ReaderException {

    r = Reader.create(URI);

    r.addTransportListener(null);

    r.connect();

    r.paramSet(TMConstants.TMR_PARAM_TAGOP_ANTENNA, 2);

```

```

ServerSocket server = new ServerSocket(RETIM_PORT);

System.out.println("ESCUTANDO A PORTA: " + RETIM_PORT);

parser = new JSONParser();

try {
    pnrđ = (JSONObject) parser.parse(new FileReader("pnrd2.json"));
    conditions = (JSONArray) pnrđ.get("conditions");
    coeficients = (JSONArray) parser.parse(new FileReader("coeficientsjson.json"));
    if (pnrd.get("outputType") != null) {
        String host = (String) pnrđ.get("host");
        int port = ((Long) pnrđ.get("port")).intValue();
        if (pnrd.get("outputType").equals("tcpip")) {
//            globalSocket = new Socket(host, port);
        } else if (pnrd.get("outputType").equals("cpn")) {
            cpn.connect(host, port);
        }
    }
} catch (ParseException ex) {

java.util.logging.Logger.getLogger(Monitor_Tags.class.getName()).log(Level.SEVERE, null,
ex);

}

try {
    while (true) {
        readTags(server);
    }
} catch (ReaderException | ParseException ex) {

java.util.logging.Logger.getLogger(Monitor_Tags.class.getName()).log(Level.SEVERE, null,
ex);

}

}

```

}

APÊNDICE B – DEMIS – IncidentMatrix

```
package continuous_adm;
```

```
/**
```

```
*
```

```
* @author Lucas
```

```
* Classe que representa um vetor de short que conterá as palavras que conterão
```

```
* na tag
```

```
*/
```

```
//CONSTRUTORES
```

```
public class IncidentMatrix {
```

```
    private ReaderWord[] palavras = null;
```

```
    private int states, transitions;
```

```
    public IncidentMatrix(short array[], int states, int transitions){        //CONSTRUTOR
RECEBENDO SHORT - DIRETO DA TAG
```

```
        setMatrix(array, states, transitions);
```

```
    }
```

```
    public IncidentMatrix(byte array[]){                                //CONSTRUTOR
RECEBENDO BYTES - CADA BYTE UM VALOR DA MATRIZ COMO É ESCRITA NO
PAPEL
```

```
        setMatrix(array);
```

```
    }
```

```
    public IncidentMatrix(byte halfWord[], int states, int transitions){    //CONSTRUTOR
RECEBENDO BYTES - CADA BYTE UM VALOR DA MATRIZ COMO É ESCRITA NO
PAPEL
```

```
        setMatrix(halfWord, states, transitions);
```

```
    }
```

```
    public IncidentMatrix(byte miniWords[], int states, int transitions, boolean whatever){
//CONSTRUTOR RECEBENDO BYTES - CADA BYTE UM VALOR DA MATRIZ
COMO É ESCRITA NO PAPEL
```

```
        setMatrix(miniWords, states, transitions, whatever);
```

```

    }

    public IncidentMatrix(ReaderWord palavras[], int states, int transitions){
//CONSTRUTOR RECEBENDO READERWORDS DIRETAMENTE

        setMatrix(palavras, states, transitions);
    }

//METODOS SET

/**
 * METODO DE TESTE, CADA BYTE UM VALOR DA MATRIZ
 * @param array
 * @return
 */
private void setMatrix(byte array[]){
    int elementsNumber = array.length;

    int wordsNumber = (int)Math.ceil((double)elementsNumber/8);

    palavras = new ReaderWord[wordsNumber];

    for(int i=0;i<wordsNumber;i++) //INSTANCIANDO CADA
OBJETO DO VETOR DE PALAVRAS
        palavras[i] = new ReaderWord();

    for(int i=0;i<elementsNumber;i++){
        if(array[i] == -1) array[i] = 2; //TROCAR -1 DA MATRIZ POR
2

        //System.out.printf("%d - %d\n", i, array[i]);

        palavras[i/8].setMiniWord(array[i], i%8);
    }
}

public void setMatrix(short array[], int states, int transitions){

    this.states = states;

    this.transitions = transitions;

    int elementsNumber = states*transitions;

    int wordsNumber = (int)Math.ceil((double)elementsNumber/8);

    palavras = new ReaderWord[wordsNumber];

```

```

    for(int i = 0;i<array.length;i++)
        palavras[i] = new ReaderWord(array[i]);
}
/**
 * METODO SET QUE TRANSFORMA 2 BYTES EM UMA PALAVRA(SHORT)
 * @param halfWord bytes que serão convertidos em ReaderWords
 * @param estados número de estados
 * @param transicoes número de transições
 */
public void setMatrix(byte halfWord[], int estados, int transicoes){
    this.states = estados;
    this.transitions = transicoes;
    int elementsNumber = estados*transicoes;
    int wordsNumber = (int)Math.ceil(((double)elementsNumber/8));
    palavras = new ReaderWord[wordsNumber];
    for(int i=0;i<wordsNumber*2;i=i+2){
        short value = (short) ((halfWord[i]<<8)+(halfWord[i+1]&0xFF));
        palavras[i/2] = new ReaderWord(value);
    }
}
/**
 * METODO SET QUE RECEBE ESTADOS*TRANSICOES MINIWORDS E AS
AGrupa
 * @param miniWords valores de cada miniWord
 * @param estados número de estados
 * @param transicoes número de transições
 */
public void setMatrix(byte miniWords[], int estados, int transicoes, boolean whatever){
    this.states = estados;
    this.transitions = transicoes;

```

```

int elementsNumber = estados*transicoes;

int wordsNumber = (int)Math.ceil((double)elementsNumber/8);

palavras = new ReaderWord[wordsNumber];

for(int i=0;i<wordsNumber;i++)                //INSTANCIANDO CADA
OBJETO DO VETOR DE PALAVRAS

    palavras[i] = new ReaderWord();

for(int i=0;i<elementsNumber;i++){

    miniWords[i] = (miniWords[i] == -1) ? 2: miniWords[i];
//TROCAR -1 DA MATRIZ POR 2

    //System.out.printf("%d - %d\n", i, array[i]);

    palavras[i/8].setMiniWord(miniWords[i], i%8);

}

}

public void setMatrix(ReaderWord palavras[], int states, int transitions){

    this.states = states;

    this.transitions = transitions;

    this.palavras = palavras;

}

//METODOS GET

public int getStatesNumber(){

    return this.states;

}

public int getTransitionsNumber(){

    return this.transitions;

}

public ReaderWord[] getWords(){

    return palavras;

}

public short[] getShorts(){

    int len = palavras.length;

    short[] buf = new short[len];

```

```

    for(int i = 0;i<len;i++)
        buf[i] = palavras[i].getWord();
    return buf;
}

public byte[] getBytes(){
    byte[] b = new byte[this.getShorts().length*2];
    int sIndex = 0;
    for(int bIndex=0;bIndex<b.length;bIndex=bIndex+2){
        b[bIndex] = (byte)((this.getShorts()[sIndex] & 0xFF00) >> 8);
        b[bIndex+1] = (byte)(this.getShorts()[sIndex] & 0x00FF);
        sIndex++;
    }
    return b;
}

/**
 * Retorna o valor correspondente(0, 1 ou 2) para tal estado e transição.
 * @param state
 * @param transition
 * @return Valor correspondente a certo estado e transição da matriz.
 */
public byte getValue(int state, int transition){
    if(state > states){
        System.out.println("NOT A VALID STATE: "+state+">"+"states);
        return -1;
    }
    else if(transition > transitions){
        System.out.println("NOT A VALID TRANSITION: "+transition+">"+"transitions);
        return -2;
    }
    int address = state*transitions+transition;

```

```

//System.out.println("ADDRESS: "+palavras.length);
byte value = palavras[address/8].getMiniWord(address%8);
if(value > 2 || value < 0)
    //throw new invalidValueException("NOT A VALID VALUE: "+ value);
    System.out.println("NOT A VALID VALUE: " + value);
return value;
}
/**
 * Retorna uma matriz de bytes correspondente a matriz incidência.@return
 */
public int[][] getMatrix(){
    int[][] incidentMatrix = new int[this.states][this.transitions];
    for(int i=0;i<this.states;i++){
        for(int j=0;j<this.transitions;j++){
            incidentMatrix[i][j] = this.getValue(i, j);
        }
    }
    return incidentMatrix;
}

public void printMatrix(){
    int[][] incidentMatrix = this.getMatrix();
    for(int i = 0;i<this.states;i++){
        for(int j=0;j<this.transitions;j++)
            System.out.print(incidentMatrix[i][j]+" ");
        System.out.println();
    }
}
}

```

APÊNDICE C – DEMIS – ReaderWord

```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

package continuous_adm;

/**
 *
 * @author Lucas
 */
public class ReaderWord {
    private short word = 0;
    public ReaderWord(){
    }
    public ReaderWord(short word){
        this.word = word;
    }
    public short getWord(){
        //System.out.printf("WORD VALUE: %X\n", this.word);
        return this.word;
    }
    public void setWordValue(short word){
        this.word = word;
    }
    public byte getMiniWord(int pos){
        return (byte)((this.word >> (14-pos*2)) & 3);
    }
}

```

```
public void setMiniWord(int value, int pos){  
    this.word &= ~(0b11 << (14-pos*2));           //ZERA A POSIÇÃO QUE  
    QUERO GRAVAR  
    this.word |= (value) << (14-pos*2);           //FAÇO OR COM O VALOR  
    DESLOCADO DA POSIÇÃO  
}  
}
```

APÊNDICE D – DEMIS – StatesVector

```

package continuous_adm;

/**
 *
 * @author Lucas
 * Classe que auxilia nas operações relacionadas aos vetores de estado
 */
public class StatesVector {
    int[] statesQuantity;

    public StatesVector(int tam){
        statesQuantity = new int[tam];
        for(int i =0;i<statesQuantity.length;i++)
            statesQuantity[i] = 0;
    }

    /**
     * Construtor a partir de uma matriz incidência e uma transição que cria
     * um vetor coluna correspondente àquela transição.
     * @param incident
     * @param transition
     */
    public StatesVector(IncidentMatrix incident, int transition){
        int b[][] = incident.getMatrix();
        statesQuantity = new int[incident.getStatesNumber()];
        for(int i=0;i<statesQuantity.length;i++)
            statesQuantity[i] = b[i][transition] == 2 ? -1 : b[i][transition];
    }

    public int getValue(int pos){
        return this.statesQuantity[pos];
    }
}

```

```

public void printVector(){
    System.out.print("VETOR: ");
    for(int i:statesQuantity)
        System.out.printf("%d ", i);
    System.out.println();
}

public void setValue(int pos, int value){
    this.statesQuantity[pos] = value;
}

/**
 * Seta a quantidade de Tags em todos os estados para 0.
 */
public void reset(){
    for(int i =0;i<statesQuantity.length;i++)
        statesQuantity[i] = 0;
}

/**
 * Soma um vetor a este.
 * @param v2 Vector a ser adicionado
 */
public void sumVector(StatesVector v2){
    if(this.statesQuantity.length != v2.statesQuantity.length){
        System.out.println("Dimensions must agree to sum the vectors");
        return;
    }
    for(int i = 0; i<statesQuantity.length;i++){
        this.statesQuantity[i]+= v2.statesQuantity[i];
    }
}

/**

```

```
* @return statesQuantity - vetor com a quantidade de tags em cada estado.  
*/  
  
public int[] getStatesQuantity(){  
    return statesQuantity;  
}  
  
}
```

APÊNDICE E – DEMIS – TagRead

```
package Extra;

import com.thingmagic.TagReadData;
import java.io.Serializable;

/**
 *
 * @author Lucas
 * Classe que simboliza uma leitura de tag. É usada para, além de simplificar
 * a organização, transmitir as leituras para os clientes que performam a
 * requisição. Desta forma, este objeto deve ser deserializado do lado do
 * cliente, ou seja, o cliente deve ter uma cópia exata deste objeto para que
 * o mesmo consiga realizar a leitura.
 */
public class TagRead implements Serializable{
    byte[] dataEPCBank;
    String EPCString;
    byte[] dataUserBank;
    byte[] EPCBytes;
    int state;
    int id;
    int rssi;
    int phase;
    int frequency;
    int antenna;

    public TagRead(TagReadData tag){
        this.dataEPCBank = tag.getEPCMemData();
    }
}
```

```

        this.dataUserBank = tag.getUserMemData();
        this.EPCString = tag.epcString();
        this.EPCBytes = tag.getTag().epcBytes();
        this.state = tag.getTag().epcBytes()[1];
        this.id = tag.getTag().epcBytes()[0];
        this.rssi = tag.getRssi();
        this.phase = tag.getPhase();
        this.frequency = tag.getFrequency();
        this.antenna = tag.getAntenna();

    }

    public TagRead(String EPCString){
        this.EPCString = EPCString;
    }

    public String getEPCString(){
        return EPCString;
    }

    public byte[] getEPC(){
        return dataEPCBank;
    }

    public byte[] getUserBank(){
        return dataUserBank;
    }

    public int getState(){
        return state;
    }

```

```
}

public int getID(){
    return id;
}

public int getRssi() {
    return rssi;
}

public int getPhase() {
    return phase;
}

public int getFrequency() {
    return frequency;
}

public int getAntenna() {
    return antenna;
}

public void setState(int state){
    this.state = state;
}
}
```

APÊNDICE F – RETIM – Visualizacao_Petri

```

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package visualizacao_petri;

import Extra.TagRead;
import com.mxgraph.model.mxCell;
import com.mxgraph.swing.mxGraphComponent;
import com.mxgraph.util.mxEvent;
import com.mxgraph.util.mxEventObject;
import com.mxgraph.util.mxEventSource;
import com.mxgraph.view.mxGraph;
import com.mxgraph.view.mxGraphSelectionModel;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Iterator;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;

```

```
import javax.swing.table.DefaultTableModel;
```

```
/**
```

```
*
```

```
* @author Felipe
```

```
*/
```

```
public class Visualizacao_Petri {
```

```
    /* Foi definida uma variável Buffer para guardar as informações sobre as ligações a serem
    desenhadas,
```

```
        * para que não se perca informações na ramificação,
```

```
        * foi definido que 7 informações armazenadas no Buffer são mais do que o suficiente
```

```
        * para guardar as informações desejadas sobre as ramificações, considerando o tamanho do
    frame.
```

```
        *
```

```
        * No Buffer a primeira transitions é para indicar o elemento que está fazendo a ligação,
    sendo definido:
```

```
        * -1 -> Buffer não utilizado,
```

```
        * 0 -> Estado,
```

```
        * 1 -> Transição
```

```
        *
```

```
        * A segunda transitions é relativa a posição do elemento na matriz.
```

```
        *
```

```
        * A terceira transitions nos dá a informação da direção de orientação do desenho, sendo
    definido:
```

```
        *
```

```
        * 0 -> Horizontal
```

```
        * 1 -> Vertical
```

```
    */
```

```
    int[][] Buffer = {{-1, -1, -1}, {-1, -1, -1}, {-1, -1, -1}, {-1, -1, -1}, {-1, -1, -1}, {-1, -1, -1},
    {-1, -1, -1}};
```

//Ligacao_Encontrada é a variavel responsável por buscar a informação no Buffer para fazer o desenho atual.

```
int[][] Ligacao_Encontrada = {{0, 0, 0}};
```

```
Resolve_Matriz m;
```

```
Estado[] Lista_Estado;
```

```
Transicao[] Lista_Transicao;
```

```
int[][] Elemento_Desenhado;
```

```
double Fator_Feedback;
```

```
mxGraph mxgraph;
```

```
mxGraphComponent graphComponent;
```

```
Object mxDefaultParent;
```

```
public Visualizacao_Petri() {
```

```
    try {
```

```
        m = new Resolve_Matriz();
```

```
    } catch (IOException ex) {
```

```
        Logger.getLogger(Visualizacao_Petri.class.getName()).log(Level.SEVERE, null, ex);
```

```
    }
```

```
}
```

```
public void Inicializa_Elemento_Desenhado() {
```

```
    if (m.transitions >= m.states) {
```

```
        Elemento_Desenhado = new int[m.transitions][2];
```

```
        for (int i = 0; i < m.transitions; i++) {
```

```
            Elemento_Desenhado[i][0] = -1;
```

```
            Elemento_Desenhado[i][1] = -1;
```

```
        }
```

```
    } else {
```

```

    Elemento_Desenhado = new int[m.states][2];
    for (int i = 0; i < m.states; i++) {
        Elemento_Desenhado[i][0] = -1;
        Elemento_Desenhado[i][1] = -1;
    }
}

}

public boolean Verifica_Elemento_Desenhado(int tipo, int posicao) {
    int controle = 0;
    for (int i = 0; i < Elemento_Desenhado.length; i++) {
        if (Elemento_Desenhado[i][tipo] == posicao && controle == 0) {
            controle++;
        }
        System.out.print(Elemento_Desenhado[i][tipo]);
    }
    if (controle != 0) {
        return true;
    } else {
        return false;
    }
}

public boolean Verifica_Token(int Token) {
    if (Token == m.state) {
        return true;
    } else {
        return false;
    }
}
}

```

```

public void Adicionar_Elemento_Desenhado(int tipo, int posicao) {
    int contador = 0;
    for (int i = 0; i < Elemento_Desenhado.length; i++) {
        if (Elemento_Desenhado[i][tipo] == -1 && contador == 0) {
            Elemento_Desenhado[i][tipo] = posicao;
            contador++;
        }
    }
}

```

```

public int[][] Procura_Buffer() {
    int[][] Posicao_Buffer = {{-1, -1, -1}};
    int controle = 0;
    for (int i = 0; i < Buffer.length; i++) {
        if (Buffer[i][0] != -1 && controle == 0) {
            Posicao_Buffer[0][0] = Buffer[i][0];
            Posicao_Buffer[0][1] = Buffer[i][1];
            Posicao_Buffer[0][2] = Buffer[i][2];

            Buffer[i][0] = -1;
            Buffer[i][1] = -1;
            Buffer[i][2] = -1;
            controle++;
        }
    }
    return Posicao_Buffer;
}

```

```

public void Coloca_No_Buffer(int[][] Dado) {

```

```

int controle = 0;
for (int i = 0; i < Buffer.length; i++) {
    if (Buffer[i][0] == -1 && controle == 0) {
        Buffer[i][0] = Dado[0][0];
        Buffer[i][1] = Dado[0][1];
        Buffer[i][2] = Dado[0][2];
        controle++;
    }
}
}

```

```

public void Iniciar_Buffer() {

    Buffer[0][0] = 0;
    Buffer[0][1] = 0;
    Buffer[0][2] = 0;

    for (int i = 1; i < Buffer.length; i++) {
        Buffer[i][0] = -1;
    }
}

```

```

public void Ver_Buffer() {
    for (int i = 0; i < Buffer.length; i++) {
        System.out.print("\n");
        for (int j = 0; j < 3; j++) {
            System.out.print(Buffer[i][j] + "\t");
        }
    }
    System.out.print("\n");
}

```

```

}

public void Mover_Elemento(Elemento E) {
    mxgraph.setSelectionCell(E.getElemento());
    mxgraph.moveCells(mxgraph.getSelectionCells(), 0, -Elemento.getTamanho() * 3);
}

public void Desenha_Ligacoes() {
    int Controle_Desenho = 0;
    int[][] Adiciona_Buffer = new int[1][3];
    int[][] matrix = m.incidentMatrix.getMatrix();
    m.statesVector.printVector();
    //System.out.println("TAMANHO MATRIX: "+matrix.length);
    // Se a Ligacao_Encontrada[0][0] == 0 faz o tratamento da ligação de um Estado
    if (Ligacao_Encontrada[0][0] == 0) {
        for (int j = 0; j < m.transitions; j++) {
            if (matrix[Ligacao_Encontrada[0][1]][j] == 2 &&
                Verifica_Elemento_Desenhado(1, j) == false) {
                Adiciona_Buffer[0][0] = 1;
                Adiciona_Buffer[0][1] = j;
                Controle_Desenho++;
                if ((Controle_Desenho == 1 && Ligacao_Encontrada[0][2] == 0) ||
                    (Controle_Desenho == 2 && Ligacao_Encontrada[0][2] == 1)) {
                    Lista_Transicao[j] = new
                    Transicao(Lista_Estado[Ligacao_Encontrada[0][1]].getX() + Elemento.getTamanho() * 2,
                                Lista_Estado[Ligacao_Encontrada[0][1]].getY(), "T" +
                                String.valueOf(j));
                    Lista_Transicao[j].Desenha_Transicao(mxgraph);
                    Desenha_Arco(Lista_Estado[Ligacao_Encontrada[0][1]], Lista_Transicao[j]);
                    Adiciona_Buffer[0][2] = 0;
                    Coloca_No_Buffer(Adiciona_Buffer);
                }
            }
        }
    }
}

```

```

    }

    if ((Controle_Desenho == 2 && Ligacao_Encontrada[0][2] == 0) ||
        (Controle_Desenho == 1 && Ligacao_Encontrada[0][2] == 1)) {

        Lista_Transicao[j] = new
        Transicao(Lista_Estado[Ligacao_Encontrada[0][1]].getX(),

            Lista_Estado[Ligacao_Encontrada[0][1]].getY() +
            Elemento.getTamanho() * 2, "T" + String.valueOf(j));

        Lista_Transicao[j].Desenha_Transicao(mxgraph);

        Desenha_Arco(Lista_Estado[Ligacao_Encontrada[0][1]], Lista_Transicao[j]);

        Adiciona_Buffer[0][2] = 1;

        Coloca_No_Buffer(Adiciona_Buffer);

    }

    Adicionar_Elemento_Desenhado(1, j);

} else {

    if (matrix[Ligacao_Encontrada[0][1]][j] == 2 &&
        Verifica_Elemento_Desenhado(1, j) == true) {

        Mover_Elemento(Lista_Estado[Ligacao_Encontrada[0][1]]);

        Desenha_Arco(Lista_Estado[Ligacao_Encontrada[0][1]], Lista_Transicao[j]);

        Fator_Feedback = Fator_Feedback + 0.5;

        System.out.println("teste 1");

    }

}

}

}

}

// Se a Ligacao_Encontrada[0][0] == 1 faz o tratamento da ligação de uma Transição
if (Ligacao_Encontrada[0][0] == 1) {
    for (int j = 0; j < m.states; j++) {

```

```

String num = String.valueOf(j);

if (matrix[j][Ligacao_Encontrada[0][1]] == 1 &&
Verifica_Elemento_Desenhado(0, j) == false) {

    Adiciona_Buffer[0][0] = 0;

    Adiciona_Buffer[0][1] = j;

    Controle_Desenho++;

    if ((Controle_Desenho == 1 && Ligacao_Encontrada[0][2] == 0) ||
(Controle_Desenho == 2 && Ligacao_Encontrada[0][2] == 1)) {

        Lista_Estado[j] = new
Estado(Lista_Transicao[Ligacao_Encontrada[0][1]].getX() + Elemento.getTamanho() * 2,

        Lista_Transicao[Ligacao_Encontrada[0][1]].getY(), "P" + num, j,
m.statesVector.getValue(j));

        Lista_Estado[j].Desenha_Estado(mxgraph, Verifica_Token(j));

        Desenha_Arco(Lista_Transicao[Ligacao_Encontrada[0][1]], Lista_Estado[j]);

        Adiciona_Buffer[0][2] = 0;

        Coloca_No_Buffer(Adiciona_Buffer);

    }

    if ((Controle_Desenho == 2 && Ligacao_Encontrada[0][2] == 0) ||
(Controle_Desenho == 1 && Ligacao_Encontrada[0][2] == 1)) {

        Lista_Estado[j] = new
Estado(Lista_Transicao[Ligacao_Encontrada[0][1]].getX(),

        Lista_Transicao[Ligacao_Encontrada[0][1]].getY() +
Elemento.getTamanho() * 2, "P" + num, j, m.statesVector.getValue(j));

        Lista_Estado[j].Desenha_Estado(mxgraph, Verifica_Token(j));

        Desenha_Arco(Lista_Transicao[Ligacao_Encontrada[0][1]], Lista_Estado[j]);

        Adiciona_Buffer[0][2] = 1;

        Coloca_No_Buffer(Adiciona_Buffer);

    }

    Adicionar_Elemento_Desenhado(0, j);

} else {

```

```

        if (matrix[j][Ligacao_Encontrada[0][1]] == 1 &&
Verifica_Elemento_Desenhado(0, j) == true) {

            Mover_Elemento(Lista_Transicao[Ligacao_Encontrada[0][1]]);

            Desenha_Arco(Lista_Transicao[Ligacao_Encontrada[0][1]], Lista_Estado[j]);

            Fator_Feedback = Fator_Feedback + 0.5;

            System.out.println("teste 2");

        }

    }

}

}

}

```

//Faz o desenho e as ligações dos estados de fora da cadeia principal

```

public void Verifica_Estados_Restantes() {

    int controle = 0;

    int matrix[][] = m.incidentMatrix.getMatrix();

    for (int j = 0; j < m.states; j++) {

        String num = String.valueOf(j);

        controle = 0;

        if (Verifica_Elemento_Desenhado(0, j) == false) {

            for (int i = 0; i < m.transitions; i++) {

                if (matrix[j][i] == 2) {

                    if (controle == 0) {

                        Lista_Estado[j] = new Estado(Lista_Transicao[i].getX(),

                            Lista_Transicao[i].getY() - Elemento.getTamanho() * 2, "P" + num, j,
m.statesVector.getValue(j));

                        Lista_Estado[j].Desenha_Estado(mxgraph, Verifica_Token(j));

                        Adicionar_Elemento_Desenhado(0, j);

                        controle++;

                    }

                }

            }

        }

    }

}

```

```

        Desenha_Arco(Lista_Estado[j], Lista_Transicao[i]);
    }
}
}
}
}

```

```

public void Desenha_Arco(Elemento e0, Elemento e1) {
    mxgraph.insertEdge(mxDefaultParent, null, "", e0.getElemento(), e1.getElemento());
}

```

```

public void Algoritmo_Rede(int posicao) {

```

```

    Iniciar_Tela();

```

```

    mxDefaultParent = mxgraph.getDefaultParent();

```

```

    Iniciar_Buffer();

```

```

    Inicializa_Elemento_Desenhado();

```

```

    Fator_Feedback = 1;

```

```

    Lista_Estado = new Estado[m.states];

```

```

    Lista_Transicao = new Transicao[m.transitions];

```

```

    Lista_Estado[0] = new Estado(Elemento.getTamanho(), posicao / 2, "P0", 0,
m.statesVector.getValue(0));

```

```

    Lista_Estado[0].Desenha_Estado(mxgraph, Verifica_Token(0));

```

```

    mxgraph.refresh();

```

```

    Adicionar_Elemento_Desenhado(0, 0);

```

```

mxgraph.getModel().beginUpdate();

while (Ligacao_Encontrada[0][0] != -1) {
    Ver_Buffer();
    Ligacao_Encontrada = Procura_Buffer();
    Desenha_Ligacoes();
}

Verifica_Estados_Restantes();

mxgraph.getModel().endUpdate();
}

public void Iniciar_Tela() {
    JFrame frame = new JFrame("RETIM (REAL TIME ITEM MONITORING)");
    frame.setSize(800, 600);
    frame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
    frame.addWindowListener(new java.awt.event.WindowAdapter(){
        @Override
        public void windowClosing(java.awt.event.WindowEvent event){ //AO FECHAR
A JANELA, FECHA O SOCKET E MANDA ORDEM DE FECHAR O SOCKET DE LÁ
            try {
                m.s.getOutputStream().write(-2);
                m.s.close();
            } catch (IOException ex) {
                Logger.getLogger(Visualizacao_Petri.class.getName()).log(Level.SEVERE, null,
ex);
            }
        }
    });
}

```

```

mxgraph = new mxGraph();
mxgraph.setCellsDisconnectable(false);
mxgraph.setCellsEditable(false);
mxgraph.setCellsResizable(false);
mxGraphSelectionModel model = mxgraph.getSelectionModel();
model.addListener(mxEvent.CHANGE, new mxEventSource.mxIEventListener() {
    @Override
    public void invoke(Object sender, mxEventObject evt) {
        mxGraphSelectionModel model = (mxGraphSelectionModel) (sender);
        ArrayList<Elemento> elementos = new ArrayList<Elemento>();
        ArrayList<Object> removed = (ArrayList) evt.getProperty("removed");
        if (removed == null) {
            return;
        }

        for (Iterator<Object> it = removed.iterator(); it.hasNext();) {
            mxCell cell = (mxCell) it.next();
            if (cell.getValue() instanceof Elemento) {
                elementos.add((Elemento) cell.getValue());
            }
        }

        model.removeListener(this);
        for (Iterator<Elemento> it = elementos.iterator(); it.hasNext();)
            it.next().updateSelection(model);
        model.addListener(mxEvent.CHANGE, this);
    }
});

graphComponent = new mxGraphComponent(mxgraph);
graphComponent.getViewport().setOpaque(false);

```

```

graphComponent.setOpaque(true);
graphComponent.setBackground(Color.WHITE);
graphComponent.setFoldingEnabled(false);
graphComponent.setConnectable(false);
frame.getContentPane().add(graphComponent);
graphComponent.getGraphControl().addMouseListener(new MouseAdapter() {
    @Override
    public void mouseReleased(MouseEvent e){
        Object cellObject = graphComponent.getCellAt(e.getX(), e.getY());
        mxCell cell = (mxCell) cellObject;
        Estado state = (Estado) cell.getValue();           //STATE QUE FOI
CLICADO
        if (cell != null && cell.getValue() instanceof Estado){
            JFrame pop = new JFrame("INFO");
            pop.setVisible(true);
            pop.setLocation(e.getX(), e.getY());
            pop.setSize(400, 200);
            pop.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
            System.out.println(cell.toString());

            JTable jTable = new javax.swing.JTable();
            DefaultTableModel model = new DefaultTableModel(
                new Object [][] { },
                new String [] {
                    "n", "EPC"
                }
            );
            int j = 1;
            for (TagRead tagRead : m.tagReads) {
                mxCell quant = (mxCell)(state.getQuantity());

```

```

        String value = String.valueOf(quant.getValue());
        if (tagRead.getState() == state.getState()) {
            model.addRow(new String[] {String.valueOf(j),
tagRead.getEPCString()});
            j++;
        }
    }
    jTable.setModel(model);
    JScrollPane scroll = new JScrollPane(jTable);
    pop.add(scroll);
}

});
frame.add(graphComponent, BorderLayout.CENTER);
frame.setVisible(true);
}
}

```

APÊNDICE G – RETIM – Resolve_Matriz

```

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package visualizacao_petri;

import Extra.TagRead;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.ObjectInputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.logging.Level;
import java.util.logging.Logger;
import visualizacao_petri.Vector;

/**
 *
 * @author Felipe
 */

/* A classe Resolve_Matriz contém apenas um método chamado Analise_String_Matriz() que
é responsável por
 * ler a informação da String e fazer a matriz incidência.
 */

public class Resolve_Matriz {

```

```

private String Matriz_inc;
int transitions;
int states;
int state;
int[][] Matriz;
int Estado_Token;
IncidentMatrix incidentMatrix;
Vector statesVector;
Socket s;
TagRead[] tagReads;
ObjectInputStream objInput;
//*****//

// Construtor da classe que pega a String de entrada, verifica se a string está em Binário e
calcula a matriz incidência

public Resolve_Matriz(String M) throws IOException {
    Matriz_inc = M;
    System.out.println(Matriz_inc);
    Analise_String_Matriz();
}

public Resolve_Matriz() throws IOException{
    this((String) null);
}

//Analisa a String e montar a matriz incidência

public void Analise_String_Matriz() throws IOException {
    this.s = new Socket("localhost", 7070);
    s.getOutputStream().write(70);
    byte[] b = new byte[30];
    objInput = new ObjectInputStream(s.getInputStream());
}

```

1ª

//REQUISICAO DE DADOS

```

        this.s.getInputStream().read(b, 0, 2);                //LE ESTADOS E
TRANSIÇÕES
        states = b[0]; transitions = b[1];

        this.s.getInputStream().read(b, 0, states*transitions); //LE MATRIZ
INCIDENCIA //recebe duas vezes ou uma vez e trata o vetor removendo o
começo?

        incidentMatrix = new IncidentMatrix(b, states, transitions);
        incidentMatrix.printMatrix();

        this.s.getInputStream().read(b, 0, states);           //LE VETOR DE ESTADOS
MAX 15

        statesVector = new Vector(b);

        System.out.println("STATES: "+states+" TRANSITIONS: "+transitions);
        statesVector.printVector();
    }
    /**
     * Método que atualiza o
     * @param s Socket que será usado na comunicação com o DEMIS
     */
    public void refreshRETIM(Socket s) throws IOException{
        byte[] b = new byte[300];

        this.s = s;

        s.getOutputStream().write(42);                       //REQUISICAO DE DADOS
REFRESH

        this.s.getInputStream().read(b, 0, states);           //LE VETOR DE ESTADOS
MAX 15 ESTADOS

        statesVector = new Vector(b);

        System.out.println("STATES: "+states+" TRANSITIONS: "+transitions);
        statesVector.printVector();

        try {
            tagReads = (TagRead[]) objInput.readObject();

            System.out.println();
        } catch (ClassNotFoundException ex) {

```

```
        Logger.getLogger(Resolve_Matriz.class.getName()).log(Level.SEVERE, null, ex);
    }
}

public void refreshRETIM() throws IOException{
    refreshRETIM(s);
}

/**
 * @return the Matriz_inc
 */
public String getMatriz_inc() {
    return Matriz_inc;
}
}
```