UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

GABRIEL TIBURSKI JÚNIOR

PetriFact: Ferramenta de Síntese e Controle de Sistemas a Eventos Discretos Baseada em Redes de Petri

Porto Alegre
2021

GABRIEL TIBURSKI JÚNIOR

PetriFact: Ferramenta de Síntese e Controle de Sistemas a Eventos Discretos Baseada em Redes de Petri

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. João Cesar Netto
Co-orientador: Prof. Dr. Marcelo Götz

Porto Alegre
2021

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Tiburski Júnior, Gabriel

PetriFact: Ferramenta de Síntese e Controle de Sistemas a Eventos Discretos Baseada em Redes de Petri / Gabriel Tiburski Júnior. – Porto Alegre: 2021.

41 f.

Orientador: João Cesar Netto

Co-orientador: Marcelo Götz

Trabalho de conclusão de curso (Graduação) – Universidade Federal do Rio Grande do Sul, Instituto de Informática. Curso de Ciência da Computação, Porto Alegre, BR–RS, 2021.

1. Sistemas a Eventos Discretos. 2. Redes de Petri. 3. FlexFact. 4. PetriFact. I. Netto, João Cesar, orient. II. Götz, Marcelo. III. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos André Bulhões
Vice-Reitora: Prof[a]. Patricia Pranke
Pró-Reitora de Ensino: Prof[a]. Cíntia Inês Boll
Diretora do Instituto de Informática: Prof[a]. Carla Maria Dal Sasso Freitas
Coordenador do Curso de Ciência de Computação: Prof. Rodrigo Machado
Bibliotecária-chefe do Instituto de Informática: Beatriz Regina Bastos Haro

Dedico este trabalho à minha família. Obrigado por me darem a oportunidade de estudar numa universidade incrível, por aguentarem o aperto da saudade e por me incentivarem durante toda essa jornada.

AGRADECIMENTOS

Agradeço à Universidade Federal do Rio Grande do Sul e a seus servidores por me proporcionarem a oportunidade de ingressar e finalizar uma graduação de altíssima qualidade em Ciência da Computação, culminando no desenvolvimento deste trabalho final. Agradeço, também, aos meus orientadores, Prof. Dr. João Cesar Netto e Prof. Dr. Marcelo Götz, por me oferecerem este desafio, além da paciência, dedicação e profissionalismo demonstrados durante o acompanhamento do meu trabalho, mostrando-se sempre disponíveis.

Registro aqui também meu muito obrigado a Francisco Knebel pela ajuda imensurável com a ferramenta LaTeX, o que tornou a escrita desta monografia muito mais suave, e à minha namorada, Eduarda Carvalho Oliveira, que, além de me apoiar incondicionalmente durante o desenvolvimento deste trabalho, também auxiliou na revisão do mesmo. Por fim, sou grato a meus colegas e amigos, especialmente aos participantes da "Turma do Pagode", que me acompanharam durante todos esses anos de graduação, trocando experiências, conhecimentos e me incentivando para que, finalmente, eu me graduasse.

RESUMO

O presente trabalho tem como objetivo o desenvolvimento de uma ferramenta didática, denominada PetriFact, capaz de modelar Redes de Petri e, a partir do modelo gerado, controlar um dado sistema flexível de manufatura. Esse sistema, que é um exemplo de Sistema a Eventos Discretos (SED), é simulado pelo software FlexFact, um simulador de planta industrial. Ambas as aplicações conseguem trocar sinais entre si através do protocolo Modbus/TCP, sinais esses que serão transformados em eventos de modo a alterar seus estados internos. O estudo de caso apresentado demonstra o funcionamento da ferramenta como um todo, sendo então utilizado como base para a discussão dos objetivos alcançados. A aplicação resultante poderá ser usada para desenvolver novos trabalhos acadêmicos ou também dentro de sala de aula, sendo mantida como um projeto de código aberto. Como trabalhos futuros, são sugeridas melhorias para a interface gráfica e experiência de usuário, a implementação de técnicas de análise e resolução de conflitos para Redes de Petri e a adição de suporte ao protocolo Simplenet.

Palavras-chave: Sistemas a Eventos Discretos. Redes de Petri. FlexFact. PetriFact.

PetriFact: A Tool for Synthesis and Control of Discrete-Event Systems Based on Petri Nets

ABSTRACT

The present work has the objective of developing an educational tool, named PetriFact, capable of modeling Petri Nets and, from the generated model, control a given flexible manufacturing system. This system, which is an example of a Discrete Event System (DES), is simulated by the software FlexFact, which is an industrial plant simulator. Both applications are able to exchange signals through the Modbus/TCP protocol, which will be transformed into events by each one of them and used to alter their own internal states. This work provides an example that demonstrates how the tool operates as a whole, being utilized as a basis to discuss the objectives achieved. The resulting application could end up being used to develop new academic works or also inside of a classroom, being maintained as an open-source project. Improvements to the graphical interface and user experience, the implementation of analysis techniques and conflict resolution for Petri Nets, and support for the Simplenet protocol are suggested as future works.

Keywords: Discrete Event Systems, Petri Nets, FlexFact, PetriFact.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CSS	Cascading Style Sheets
HTML	HyperText Markup Language
IDE	Integrated Development Environment
IOPT	Input/Output Place/Transition
IPC	Inter-Process Communication
SED	Sistema a Eventos Discretos
SVG	Scalable Vector Graphics
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
XML	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação e Objetivo

Conforme a literatura, sistemas flexíveis de manufatura são uma das aplicações mais extensivamente usadas e mais bem sucedidas de Redes de Petri (ZURAWSKI; ZHOU, 1995). Isso provavelmente se deve ao fato de que Redes de Petri são modelos matemáticos capazes de modelar sistemas assíncronos e concorrentes (PETERSON, 1977), características muito comuns nesses tipos de sistemas (ou em sistemas a eventos discretos, em geral).

Atualmente, durante o curso de Engenharia de Controle e Automação oferecido pela Escola de Engenharia da Universidade Federal do Rio Grande do Sul (UFRGS), é utilizado em sala de aula o software FlexFact (FGDES, 2011), uma ferramenta capaz de modelar e simular sistemas flexíveis de manufatura virtualmente, a fim de ensinar alunos sobre automação. Como complemento, também é utilizado o software DESTool (FGDES, 2008), capaz de atuar como controlador de um sistema de manufatura a partir de uma abordagem baseada em autômatos finitos.

Como objetivo, o autor pretende desenvolver uma ferramenta didática como alternativa ao software DESTool. Essa ferramenta também deve ser capaz de atuar como controladora de um sistema modelado dentro do software FlexFact, mas deve ser baseada em Redes de Petri que, conforme citado anteriormente, demonstra-se como uma ótima abordagem para esse tipo de sistema. Essa ferramenta será denominada PetriFact.

1.2 Requisitos

Para ser considerada bem-sucedida, a aplicação PetriFact deve atender aos seguintes requisitos:

1. Deve apresentar uma interface (preferencialmente gráfica) que permita ao usuário modelar Redes de Petri, instanciando lugares e transições na tela, e conectando arcos entre esses elementos (que representarão pré e pós-condições);
2. Deve conseguir comunicar-se com o simulador FlexFact através de qualquer protocolo de comunicação (desde que suportado por este último);
3. Deve interpretar os eventos (ou sinais) oriundos do simulador FlexFact de modo

que eles tenham impacto sobre quais transições devem ser disparadas;

4. Deve ser capaz de calcular e decidir, com base no estado atual da Rede de Petri e dos eventos recebidos, qual será o próximo estado da rede e quais eventos devem ser disparados para o simulador FlexFact;

1.3 Estrutura do trabalho

O presente trabalho é dividido em 5 capítulos, sendo este, "Introdução", o primeiro deles. A seguir, no capítulo "Fundamentação Teórica e Ferramentas", serão abordados conceitos-chave para o entendimento do trabalho e de seu objetivo como um todo. O terceiro capítulo, "Desenvolvimento", guiará o leitor através dos detalhes de implementação da ferramenta desenvolvida durante este trabalho. "Resultados", o quarto capítulo, apresentará o funcionamento da ferramenta em conjunto com uma análise dos objetivos alcançados. Finalmente, na "Conclusão", será discutido um breve resumo do trabalho, aplicabilidade e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA E FERRAMENTAS

As próximas seções descreverão conceitos-chave para o entendimento da proposta deste trabalho. Elas explicarão o que são sistemas a eventos discretos, apresentará o simulador FlexFact, como funciona e qual seu objetivo e, finalmente, definirá o que são Redes de Petri, provendo alguns exemplos a fim de contextualizar o leitor.

2.1 Sistemas a Eventos Discretos

De modo simples, sistemas são conjuntos de elementos relacionados que formam um todo, capazes de comunicar-se com o exterior através de suas fronteiras. Eventos discretos, por sua vez, são estímulos que representam mudanças no estado interno de um sistema, acontecendo abruptamente em determinado instante de tempo (CURY, 2001).

Sistemas, em geral, podem ser classificados como orientados pelo tempo ou a eventos. Sistemas orientados pelo tempo dependem de um relógio onde, a cada ciclo, um ou mais eventos quaisquer podem ser selecionados (desde que pertencentes ao conjunto possível de eventos daquele sistema). A partir destes, o sistema será capaz de atualizar seu próprio estado interno. Ao mesmo tempo, existe a possibilidade de nenhum evento ser selecionado e, consequentemente, nenhuma mudança de estado ocorrer. Sistemas orientados a eventos, por sua vez, não dependem de um ciclo de relógio: os eventos ocorridos são capazes de anunciarem-se para o sistema durante instantes de tempo irregulares e desconhecidos. Por definição, um Sistema a Eventos Discretos é um sistema com estados discretos, orientado a eventos, onde a evolução de seu estado depende inteiramente da ocorrência de eventos discretos assíncronos (CASSANDRAS; LAFORTUNE, 2008).

2.2 FlexFact

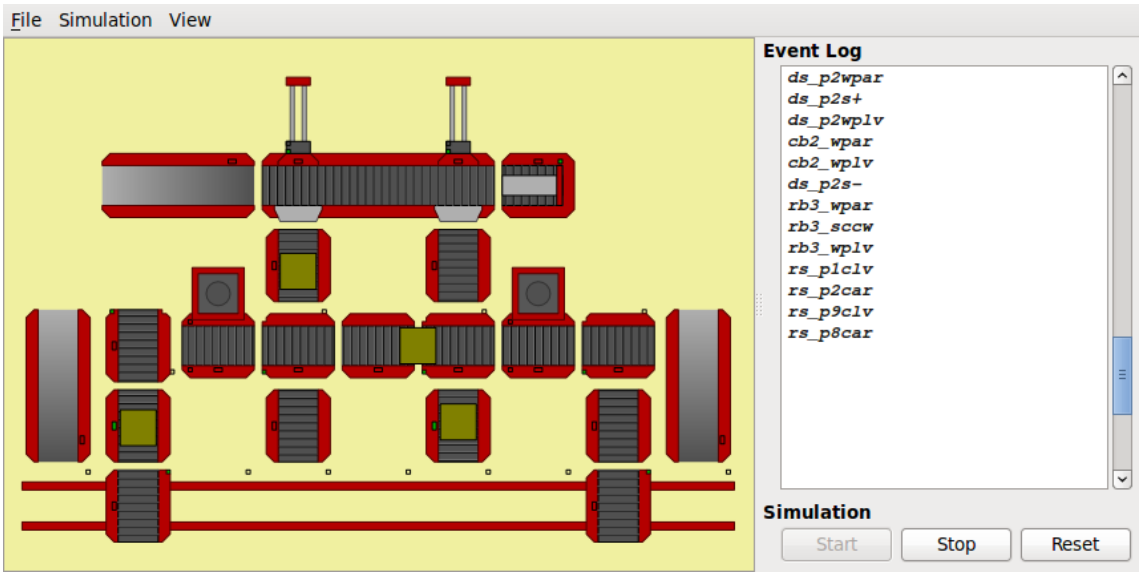
FlexFact é um simulador de planta industrial desenvolvido pelo Discrete Event Systems Group da Universidade de Erlangen-Nuremberga (Friedrich-Alexander-Universität Erlangen-Nürnberg), localizada na Alemanha (FGDES, 2011). Sua construção foi baseada em um modelo de sistema de manufatura flexível físico proposto por Hans Reger e Klaus Schmidt em 2003 e é montado e disponibilizado pela empresa Staudinger GmbH, também alemã. Sua versão virtual, denominada FlexFact, surgiu a partir

da necessidade de modelar casos mais complexos, onde o custo final pudesse ser um fator limitante.

O simulador replica o comportamento de uma série de componentes eletromecânicos, tais como esteiras transportadoras e máquinas processadoras, os quais podem ser criados e manipulados pelo usuário, permitindo-o construir o *layout* de uma linha de produção à sua vontade.

Além disso, FlexFact também é capaz simular o funcionamento desses componentes de forma visual, fornecendo interfaces para que algum controlador externo (ou até mesmo o próprio usuário) possa interagir com o sistema através de eventos de entrada e saída. Exemplos de controladores externos podem ser controladores lógico programáveis, ou até mesmo a ferramenta proprietária DESTool (FGDES, 2008), também desenvolvida pelo Discrete Event Systems Group, baseada em autômatos finitos. Com isso, é possível inferir que o software FlexFact é, em essência, um simulador de sistemas a eventos discretos. Na Figura 2.1, é possível ver o simulador FlexFact em ação, movimentando pacotes entre esteiras e registrando eventos na tela.

Figura 2.1 – Captura de tela da aplicação FlexFact



Fonte: https://fgdes.tf.fau.de/flexfact.html

Nas subseções seguintes será explicado, resumidamente, o funcionamento de diversos aspectos do simulador FlexFact, como a interface gráfica, a interação do usuário com a ferramenta, a modelagem de sistemas fabris, a simulação de sinais e eventos discretos e a comunicação com agentes externos. Para informações mais detalhadas, é sugerido ao leitor consultar o site oficial da ferramenta.

2.2.1 Modelagem de sistema fabril

Através de uma interface gráfica, a ferramenta fornece ao usuário a possibilidade de criar, manipular e visualizar modelos de sistemas fabris que podem assemelhar-se a cenários encontrados no mundo real. Para possibilitar isso, é fornecido um quadro em branco e um conjunto de componentes pré-definidos que, em conjunto, irão compor o referido modelo. A última peça restante para possibilitar a simulação do sistema é a chamada "peça de trabalho" (*workpiece*, em inglês). Por conveniência, o autor irá se referir a ela como "pacote" daqui em diante. Esses pacotes são representados como quadrados amarelos dentro da ferramenta, e seu propósito é atuar como um produto dentro do sistema fabril. Ou seja, ele será transportado, processado e entregue pelos componentes. Dentre os componentes disponíveis, podemos citar:

- Unidade alimentadora (*Stack Feeder*): a unidade alimentadora é responsável por receber novos pacotes e distribuí-los em sequência para o próximo componente da cadeia. A inserção de novos pacotes no sistema é feita de forma manual pelo clique do mouse pelo usuário;
- Esteiras: São responsáveis por movimentar pacotes entre um componente e outro. São fundamentais para o modelo, e estão presentes em boa parte do acervo de componentes de forma embutida;
- Máquinas de processamento: Além de possuírem esteiras, esses componentes também possuem algum tipo de maquinário que possibilita ao modelo simular formas de "processar" os pacotes que passam por eles, até mesmo simulando falhas;
- Painel do operador: Painel que disponibiliza múltiplos botões configuráveis que permitem ao usuário interagir de forma fácil com o sistema;
- Escorregador de saída (*Exit Slide*): Normalmente, o destino de um pacote será um escorregador de saída, que armazenará pacotes em fila até ser, enfim, limpo (parcial ou completamente). Essa limpeza ocorre a partir da interação do usuário através do clique do mouse;

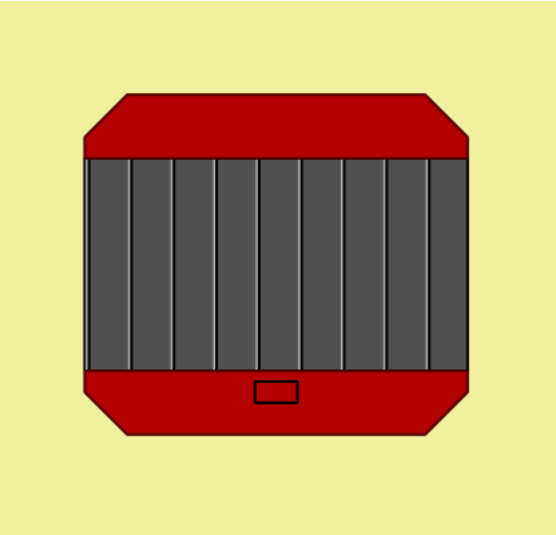
A lista completa de componentes disponíveis na ferramenta pode ser encontrada no website do desenvolvedor em https://fgdes.tf.fau.de/flexfact.html.

2.2.2 Simulação de eventos discretos

Os componentes disponíveis no simulador FlexFact são capazes de apresentar diferentes tipos de comportamento. Alguns exemplos disso são movimentar esteiras para frente e para trás, girar ao redor do próprio eixo e empurrar um pacote em determinada direção. Para tornar isso possível, cada componente possui um conjunto de sinais de entrada. Cada comportamento é ativado (ou não) a partir do resultado de uma função cujos argumentos pertencem a esse conjunto de sinais.

A fim de exemplificar o parágrafo acima, considere o componente mais simples presente no simulador até então: o *Conveyor Belt*. Este componente é representado por uma esteira, cujos sinais de entrada são apresentados na Tabela 2.1.

Figura 2.2 – Representação gráfica do componente *Conveyor Belt*



Fonte: Autor

Tabela 2.1 – Sinais de entrada do componente *Conveyor Belt*

Sinal	*Descrição*
CB_BM+	Esteira deve movimentar-se no sentido leste
CB_BM-	Esteira deve movimentar-se no sentido oeste

Fonte: https://fgdes.tf.fau.de/flexfact.html

Sinais, dentro do simulador FlexFact, podem ser representados por valores booleanos (verdadeiro ou falso, 1 ou 0) que indicarão se o sinal está ligado ou desligado. Considerando os sinais de entrada do componente *Conveyor Belt*, descritos acima, existem quatro possíveis combinações de sinais que a ferramenta será capaz de interpretar de

modo a decidir se deve ou não executar determinado comportamento. Essas combinações são descritas pela Tabela 2.2.

Tabela 2.2 – Combinações de sinais de entrada do componente *Conveyor Belt*

CB_BM+	*CB_BM-*	*Resultado esperado*
0	0	Esteira mantêm-se parada
1	0	Esteira movimenta-se no sentido leste
0	1	Esteira movimenta-se no sentido oeste
1	1	Esteira mantêm-se parada

Fonte: Autor

Os sinais de saída funcionam de forma similar aos de entrada. Ambos são representados por valores booleanos, mas, ao invés de ditar o comportamento que o componente deve apresentar, os sinais de saída são responsáveis por refletir o estado atual do sistema para agentes externos. No que se refere ao componente *Conveyor Belt*, por conta deste ser muito simples, ele possui apenas um sinal de saída. Este sinal está descrito na Tabela 2.3.

Tabela 2.3 – Sinais de saída do componente Conveyor Belt

Sinal	*Descrição*
CB_WPS	Ligado caso algum pacote esteja ao centro da esteira, desligado caso contrário

Fonte: https://fgdes.tf.fau.de/flexfact.html

Embora os sinais de saída e de entrada sejam suficientes para o gerenciamento do sistema como um todo, o simulador FlexFact também oferece a possibilidade de uso de eventos discretos ao invés de sinais. Podemos considerar esses eventos como uma forma de abstração construída por cima dos sinais elementares da ferramenta. Da mesma forma que os sinais, é possível categorizar os eventos como de entrada e de saída.

Os eventos de entrada são responsáveis por garantir que um determinado comportamento será refletido no sistema. Para isso, ao receber um evento de entrada de um agente externo, o simulador FlexFact é responsável por interpretá-lo e fazer os devidos ajustes aos sinais de entrada relevantes. Analisando novamente a Tabela 2.2, é possível visualizar que existem apenas três resultados possíveis: mover a esteira para leste, oeste, ou pará-la completamente. Os mesmos resultados podem ser alcançados com o uso dos eventos de entrada descritos pela Tabela 2.4.

Tabela 2.4 – Eventos de entrada do componente *Conveyor Belt*

Evento	*Ações necessárias*	*Resultado esperado*
cb_bm+	Ativar CB_BM+ e desativar CB_BM-	Esteira deve mover-se em sentido oeste
cb_bm-	Ativar CB_BM- e desativar CB_BM+	Esteira deve mover-se em sentido leste
cb_boff	Desativar CB_BM+ e CB_BM-	Esteira deve parar

Fonte: https://fgdes.tf.fau.de/flexfact.html

Os eventos de saída, por sua vez, são emitidos pelo próprio sistema e ocorrem apenas quando há variações de sinais específicos. No caso do componente *Conveyor Belt*, apenas um sinal de saída existe: aquele que informa se um pacote chegou ao centro da esteira ou não. Com isso, é possível inferir que existam dois eventos distintos a partir dessa informação, ambos descritos na Tabela 2.5.

Tabela 2.5 – Eventos de saída do componente *Conveyor Belt*

Evento	*Disparado por*	*Descrição*
cb_wpar	CB_WPS variou de 0 para 1	Pacote chegou até o centro da esteira
cb_wplv	CB_WPS variou de 1 para 0	Pacote deixou o centro da esteira

Fonte: https://fgdes.tf.fau.de/flexfact.html

O conjunto de sinais de entrada, sinais de saída, eventos de entrada e eventos de saída são as ferramentas disponibilizadas pelo simulador para que agentes externos possam interpretar e controlar o funcionamento de um sistema completo modelado dentro do próprio FlexFact.

2.2.3 Comunicação com agentes externos

O simulador FlexFact disponibiliza três diferentes alternativas para interação com o sistema controlador. Essas alternativas são:

- Interface gráfica: A solução mais intuitiva para visualização do estado do sistema e interação com seus componentes com certeza é a interface gráfica. O simulador apresenta na tela não apenas os sinais e os eventos disparados, mas também uma representação do modelo em tempo real. Ao mesmo tempo, ele permite ao usuário interagir com as diversas funcionalidades de cada componente com um simples clique do mouse;
- Interface Modbus/TCP: A execução do simulador através de uma interface utili-

zando o protocolo Modbus/TCP (MODBUS ORGANIZATION, 2012) possibilita a um agente externo comunicar-se com a ferramenta através da leitura e escrita de sinais, sejam eles individuais ou em lote. O simulador assume o papel de servidor (escravo), aguardando a conexão de algum cliente (mestre) pela porta 1502, usando TCP. É de responsabilidade do cliente, atuando como mestre, iniciar a conversa e saber quais sinais devem ser lidos ou escritos. Além disso, o servidor também não emite nenhum tipo de mensagem de forma autônoma, ou seja, o cliente deverá consultar o servidor periodicamente para manter sua cópia do estado do sistema atualizada. De forma a facilitar a integração de um agente externo com o modelo, o simulador FlexFact provê uma forma de exportar sua configuração interna de modo que um cliente Modbus/TCP seja capaz de interpretá-la e interagir com o sistema. Esse arquivo descreve todos os sinais presentes no modelo e como convertê-los para eventos, além de informações para realização da conexão, como número da porta a ser utilizada.

- Interface Simplenet: Diferentemente da interface Modbus/TCP, a interface Simplenet trabalha com eventos discretos ao invés de sinais. Essa interface é capaz de encontrar agentes externos participantes do sistema através de *broadcasts* UDP pela porta 40000. Esse reconhecimento também possui um modo que serve de *fallback* em caso de falha, chamado de "Simplenet (Local)". Esse modo fará com que o simulador encontre apenas agentes presentes na mesma máquina. Concluída a etapa de configuração, o sistema é capaz de emitir e receber sinais de todos os agentes presentes no sistema. Isso quer dizer que não é necessário uma leitura periódica do sistema pelos agentes, pois o próprio FlexFact se encarregará de emitir os eventos de saída a todos, enquanto os agentes ficarão encarregados de emitir os eventos de entrada para o FlexFact. Da mesma forma que a interface Modbus/TCP, o simulador também disponibiliza um arquivo de configuração Simplenet para facilitar a integração dos agentes.

2.3 Redes de Petri

2.3.1 Representação gráfica

Redes de Petri podem ser graficamente representadas como um grafo direcionado, onde arcos conectam-se a dois tipos distintos de nós: lugares e transições.

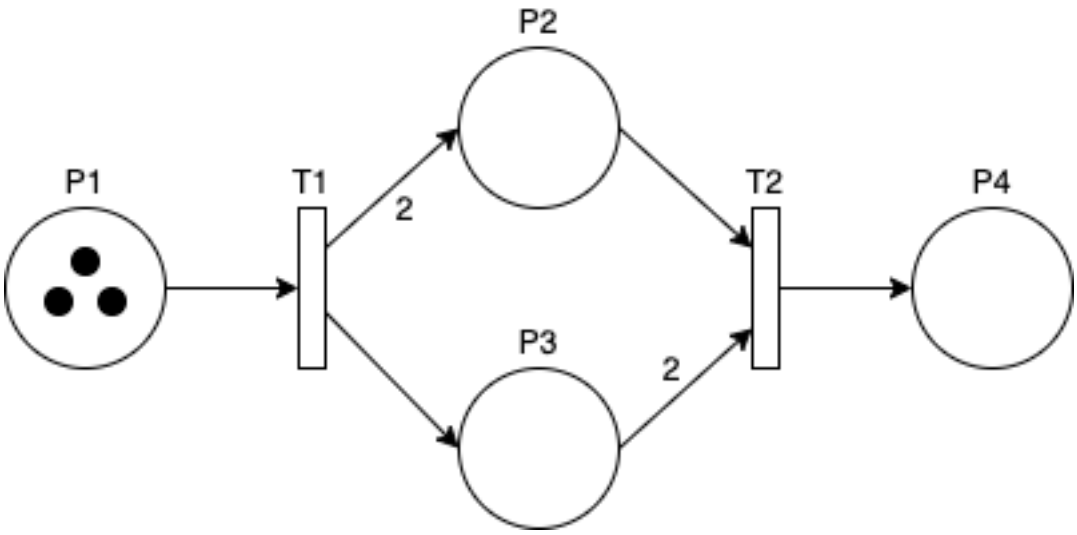
Lugares, geralmente, são representados por círculos, que podem conter uma marcação com um número discreto positivo em seu interior. Esse número também pode ser considerado como a quantidade de *tokens* armazenados pelo lugar. Transições, por sua vez, são representadas por barras ou retângulos.

Arcos são representados por setas direcionadas e devem, obrigatoriamente, conectar um lugar a uma transição e vice-versa. Ou seja, as únicas combinações possíveis de arcos são:

- De um lugar para uma transição (também chamados de pré-condições);
- De uma transição para um lugar (também chamados de pós-condições);

Além disso, arcos podem também possuir um valor associado que representa o seu peso (inteiro e positivo), tornando-se Redes de Petri não-ordinárias. Esses pesos serão usados para definir quantos *tokens* uma pré-condição consumirá ou uma pós-condição gerará. Caso omitido, esse valor deve ser considerado como um. A Figura 2.3 apresenta um exemplo de uma Rede de Petri com lugares, transições, arcos, marcações e pesos.

Figura 2.3 – Exemplo de uma Rede de Petri



Fonte: Autor

2.3.2 Representação matemática

Redes de Petri são representadas de diversas maneiras pela literatura. Considerando o trabalho de Peterson (1981), uma Rede de Petri marcada e com pesos pode ser

formalizada como uma tupla (P, T, W, M_0), onde:

- P representa um conjunto finito de lugares;
- T representa um conjunto finito de transições;
- P e T são conjuntos disjuntos, ou seja, não possuem elementos comuns entre si;
- W representa um conjunto finito de arcos juntamente com seus pesos associados, onde $W = (P \times T) \cup (T \times P) \rightarrow \mathbb{N}^+$;
- M representa uma associação de cada lugar com sua respectiva marcação (número de *tokens*), onde $M = P \rightarrow \mathbb{N}$;
- M_0 representa o conjunto inicial de marcações.

Como exemplo, a definição formal da Rede de Petri apresentada na Figura 2.3 seria composta pela tupla (P, T, W, M_0), onde:

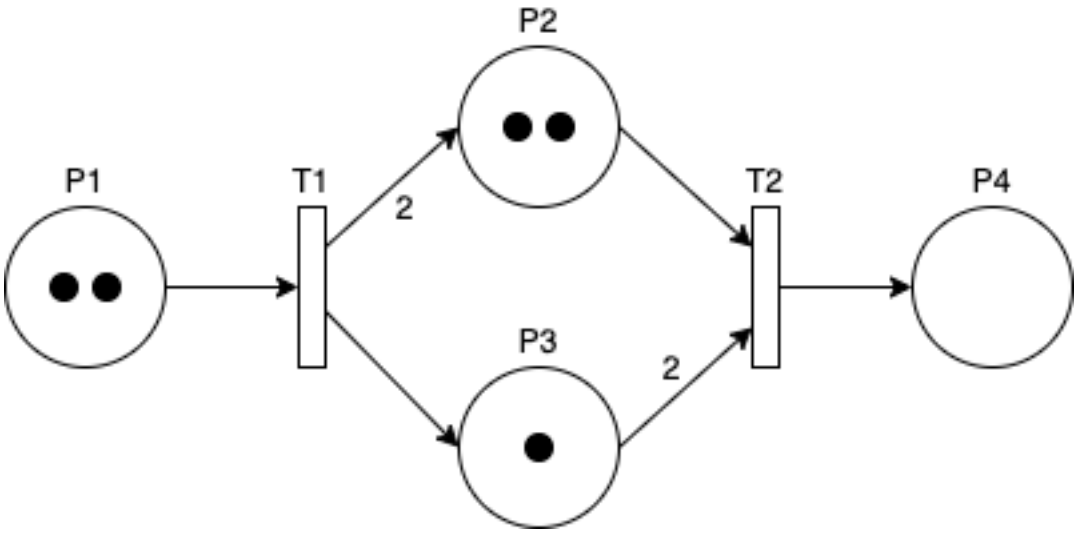
$$\begin{aligned} P &= \{P1, P2, P3, P4\} \\ T &= \{T1, T2\} \\ W &= \{(P1, T1) \rightarrow 1, (T1, P2) \rightarrow 2, (T1, P3) \rightarrow 1, \\ &\quad (P2, T2) \rightarrow 1, (P3, T2) \rightarrow 2, (T2, P4) \rightarrow 1\} \\ M_0 &= \{P1 \rightarrow 3, P2 \rightarrow 0, P3 \rightarrow 0, P4 \rightarrow 0\} \end{aligned} \tag{2.1}$$

2.3.3 Semântica de execução

Em Redes de Petri, *tokens* atuam como peças em um jogo de tabuleiro: quando uma transição é disparada, *tokens* devem ser removidos dos lugares que atuam como pré-condições da transição e, então, adicionadas aos lugares que atuam como pós-condições dessa mesma transição. Uma transição só pode ser disparada caso ela esteja ativa, ou seja, todas as suas pré-condições são atendidas.

No exemplo da Figura 2.3, é possível observar que apenas a transição T1 está ativa. Essa transição T1 requer que o lugar P1 possua ao menos um token. Caso a transição T1 seja disparada, P1 perderá um token e, de acordo com os arcos que partem de T1, P2 recerá dois tokens e P3 receberá um token. O resultado dessa operação pode ser visualizado na Figura 2.4.

Figura 2.4 – Rede de Petri da Figura 2.3 após disparo de T1



Fonte: Autor

2.3.4 Forma matricial

Para fins de execução, Redes de Petri também podem ser representadas por sua forma matricial, muito semelhante à representação matemática apresentada anteriormente.

O conjunto de pré-condições, isto é, arcos que partem de lugares até transições, pode ser representado como uma matriz de tamanho $|T| \times |P|$, denominada matriz W_-, onde cada elemento $(i \times j)$ da matriz representará o peso do arco partindo do lugar j até a transição i. Caso dois elementos não sejam conectados por um arco, o valor da célula será 0.

De forma análoga, o conjunto de pós-condições também pode ser representado por uma matriz de tamanho $|T| \times |P|$, dessa vez denominada W_+. Neste caso, cada elemento $(i \times j)$ da matriz representará o peso do arco partindo da transição i até o lugar j.

A diferença entre essas duas matrizes é definida como $W_T = -(W_-) + (W_+)$.

Finalmente, as marcações da rede são representadas como um vetor de tamanho $|P|$, onde cada elemento do vetor representa a quantidade de *tokens* presentes em cada lugar.

As definições acima, juntamente com um conjunto de transições a serem disparadas, possibilitam calcular o novo conjunto de marcações da Rede de Petri após o disparo

das transições informadas. Essa equação pode ser definida como:

$$M_{n+1} = (D_n \times W_T) + M_n \tag{2.2}$$

Na equação acima, D_n é um conjunto de tamanho $|T|$, onde $D_n(i)$ deve ser 1 caso a transição deva ser disparada, e 0 caso contrário. M_n é o conjunto de marcações atual. M_{n+1} será o conjunto de marcações no próximo instante.

Considerando a forma matricial descrita nesta seção, a Rede de Petri apresentada na Figura 2.3 seria especificada da seguinte maneira:

$$\begin{gathered} W_- = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \end{bmatrix}, W_+ = \begin{bmatrix} 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ W_T = -\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 2 & 1 & 0 \\ 0 & -1 & -2 & 1 \end{bmatrix} \\ M_0 = \begin{bmatrix} 3 & 0 & 0 & 0 \end{bmatrix} \end{gathered} \tag{2.3}$$

E, após o disparo da transição T1, as próximas marcações da rede podem ser calculadas da seguinte maneira:

$$\begin{gathered} M_1 = (D_0 \times W_T) + M_0 \\ M_1 = (\begin{bmatrix} 1 & 0 \end{bmatrix} \times \begin{bmatrix} -1 & 2 & 1 & 0 \\ 0 & -1 & -2 & 1 \end{bmatrix}) + \begin{bmatrix} 3 & 0 & 0 & 0 \end{bmatrix} \\ M_1 = \begin{bmatrix} -1 & 2 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 3 & 0 & 0 & 0 \end{bmatrix} \\ M_1 = \begin{bmatrix} 2 & 2 & 1 & 0 \end{bmatrix} \end{gathered} \tag{2.4}$$

Ao comparar o valor de M_1 com a Figura 2.4, é possível constatar que o resultado obtido pela definição da Seção 2.3.3 foi o mesmo obtido pela forma matricial.

3 DESENVOLVIMENTO

Neste capítulo serão apresentadas as técnicas e ferramentas utilizadas neste trabalho, assim como as decisões por trás das escolhas e quais foram seus impactos. Além disso, será explicado também o funcionamento dos componentes do sistema e a arquitetura por trás da aplicação como um todo.

3.1 Arquitetura do sistema

Como o simulador FlexFact é um sistema multi-plataforma, disponibilizado para os sistemas operacionais Windows, Linux e MacOS, foi de interesse dos envolvidos neste trabalho encontrar uma ferramenta que fosse versátil o suficiente para compilar e gerar executáveis para estes mesmos sistemas operacionais de forma descomplicada. A solução que atendeu da melhor maneira essa necessidade foi o *framework* Electron (OPENJS FOUNDATION, 2013), que é um projeto de código-aberto que permite o desenvolvimento de aplicações *desktop* multi-plataforma utilizando tecnologias muito similares às que são encontradas no desenvolvimento web moderno, tais como JavaScript, HTML e CSS.

De modo a possibilitar o uso dessas tecnologias independentemente do sistema operacional do usuário, Electron faz uso do Chromium, que é um navegador multiplataforma de código-aberto desenvolvido pela Google (GOOGLE, 2008). Isso possibilita ao *framework* renderizar páginas web como parte da interface gráfica do sistema.

Uma das limitações de se utilizar tecnologias empregadas no desenvolvimento web é que a especificação do JavaScript usada nesse contexto não é capaz de atender à necessidades comuns de aplicações *desktop* por questões de segurança, não fornecendo uma interface capaz de interagir diretamente com o sistema operacional do usuário (como, por exemplo, ter total controle sobre leitura e escrita de arquivos e *sockets*). Como forma de contornar essa limitação, o *framework* Electron também inclui o software NodeJS (OPENJS FOUNDATION, 2009), que será o responsável por suprir esse tipo de funcionalidade à aplicação.

De forma resumida, o funcionamento do *framework* Electron ocorre da seguinte forma: Ao iniciar a aplicação, por meio de um arquivo executável ou linha de comando, o processo principal é criado. Por si só, esse processo não é capaz de apresentar ao usuário uma interface gráfica. Sua responsabilidade principal é de configurar e criar novos pro-

cessos, denominados processo de renderização, que serão os responsáveis por prover ao usuário a parcela visual e interativa da aplicação. Cada processo de renderização normalmente é associado a uma janela nova no sistema operacional, e cada um deles equivale a uma página web de um navegador convencional, executando a pilha de tecnologias já mencionada anteriormente: HTML, CSS e JavaScript.

Além disso, o processo principal também é responsável por orquestrar a comunicação entre os processos de renderização, proporcionando uma forma de comunicação entre processos (IPC) baseada em emissões de eventos, que podem ser síncronos ou assíncronos. Cada evento despachado deve conter um nome, que atuará como identificador do evento, e um objeto qualquer que respeite as especificações da linguagem JavaScript.

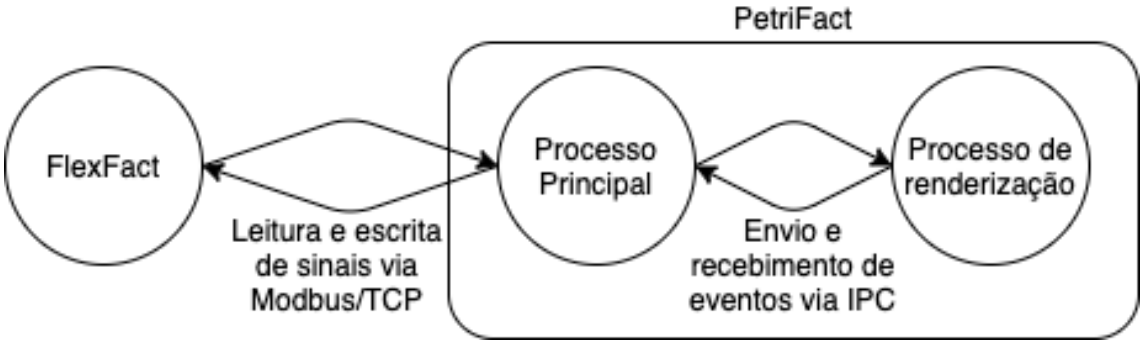
Como linguagem de programação principal, o autor optou por um subconjunto do JavaScript, denominado TypeScript (BIERMAN; ABADI; TORGERSEN, 2014). O motivo disso é devido ao fato de que JavaScript é uma linguagem dinâmica, que infere tipos automaticamente, e que não realiza verificações estritas que garantam que o código desenvolvido não terá problemas de sintaxe ou tipagem em tempo de execução. Mesmo que TypeScript apenas amenize e não resolva completamente essa última propriedade, essa linguagem foi escolhida pois, além de possuir a capacidade de definir tipos explicitamente, também oferece um conjunto maior de estruturas, como classes, interfaces e módulos, sem perder compatibilidade com as bibliotecas existentes. Isso faz de TypeScript uma linguagem mais escalável, o que pode também contribuir na produtividade do desenvolvedor ao trabalhar em conjunto com uma IDE, facilitando a detecção de erros e a sugestão de nomes pela ferramenta.

Após escolher a ferramenta mais adequada para a construção da aplicação, foi necessário também decidir como seria feita a comunicação com o simulador FlexFact. Haviam duas opções disponíveis: utilizar o protocolo Modbus/TCP ou, então, o protocolo Simplenet. Apesar de oferecer um protocolo que se adequa mais à nossa proposta do que o Modbus/TCP, a grande desvantagem do protocolo Simplenet foi a falta de bibliotecas pré-existentes disponíveis para se trabalhar em JavaScript, muito provavelmente devido ao fato dele ser proprietário. Como forma de evitar despender tempo com a construção de uma biblioteca dedicada para o uso do protocolo Simplenet, o protocolo Modbus/TCP mostrou-se mais atraente e foi o escolhido como forma de comunicação entre as aplicações.

Por fim, a arquitetura do sistema pode ser visualizada na Figura 3.1, onde cada componente do sistema é representado por um círculo, e cada seta representa formas de

comunicação entre os componentes existentes.

Figura 3.1 – Arquitetura do sistema PetriFact



Fonte: Autor

3.2 Interface gráfica

Considerando a arquitetura proposta para o sistema na seção anterior, a interface gráfica ficaria sob responsabilidade do processo de renderização. Como forma de auxiliar no processo de desenvolvimento e organização do código gerado, a primeira grande decisão no quesito interface gráfica foi a adoção de React pelo projeto. React é uma biblioteca JavaScript para criar interfaces de usuário baseadas em componentes declarativos e reutilizáveis (FACEBOOK OPEN SOURCE, 2013).

Componentes em React podem ser escritos como funções (também chamados de *"Function Components"*) que recebem um conjunto de argumentos. Cada componente, baseado em sua lógica interna, pode ser responsável por instanciar ainda mais componentes, o que, ao final da renderização da página, acaba resultando em uma árvore de componentes virtuais. Esses componentes são autogerenciáveis no sentido de que são capazes de, inteligentemente, descobrir quando devem atualizar a sua representação gráfica na tela (e de seus filhos), e também quando devem remover-se da árvore.

Como recurso para facilitar a configuração do ambiente de desenvolvimento como um todo, incluindo o *framework* Electron, a biblioteca React, a possibilidade de usar TypeScript, além de algumas ferramentas focadas na produtividade do desenvolvedor, o projeto base foi criado a partir do modelo gerado pelo software "Electron React Boilerplate" (ELECTRON REACT BOILERPLATE, 2015).

De forma a auxiliar o gerenciamento do estado da aplicação, foram utilizadas tam-

bém as bibliotecas Redux e Redux-Saga. A biblioteca Redux é responsável por gerenciar um estado global da aplicação que pode ser facilmente alterado por qualquer componente a partir do despacho de ações. Essas ações então passam por funções redutoras, que são responsáveis por atualizar o estado da aplicação. Finalmente, esse estado é disponibilizado para os componentes que, como estão estruturados em forma de árvore, acabam cascateando o estado global para todos os seus filhos e, consequentemente, para a árvore inteira. Os componentes, então, serão capazes de interpretar esses dados de modo a descobrir se devem atualizar a sua representação gráfica ou não (ABRAMOV, 2015).

Já a biblioteca Redux-Saga é uma extensão sobre Redux, e é utilizada principalmente para processos que são, por natureza, assíncronos. Um exemplo disso seria a comunicação entre o processo de renderização e o processo principal, onde seria criada uma "saga" que descreveria essa funcionalidade, disparando ações Redux conforme a "saga" é executada. (ELOUAFI, 2015).

Com isso, a fundação para a criação de uma interface gráfica para a aplicação proposta está completa. A próxima etapa do projeto consiste em utilizar essa fundação para criar uma das peças fundamentais do sistema: a habilidade de modelar uma Rede de Petri.

3.3 Modelagem da Rede de Petri

A aplicação PetriFact deve permitir que o usuário seja capaz de modelar Redes de Petri conforme a definição descrita no capítulo de "Fundamentação Teórica e Ferramentas"deste trabalho. Como descrito nos objetivos deste trabalho, uma interface gráfica dedicada é um dos requisitos básicos para a aplicação. Para atender esse requisito, foi necessário desenvolver formas de criar, editar, mover e remover elementos que representem lugares, transições e arcos.

Foram avaliadas diversas soluções para representação e manipulação de elementos gráficos compatíveis com a nossa arquitetura. Dentre as soluções avaliadas, a que mais atendeu às necessidades da aplicação proposta foi a biblioteca de código aberto JointJS, que tem como foco a criação de diagramas interativos complexos utilizando-se de gráficos vetorizados (SVGs) (CLIENT.IO, 2013).

A utilização da biblioteca JointJS é simples: basta instanciar um objeto do tipo "paper" referenciando algum bloco existente na interface, informar os atributos desejados, como cor de fundo, largura e altura, e a biblioteca tomará conta de configurar um "quadro

branco" onde os elementos da Rede de Petri viverão. Vinculado a este "paper", deve ser criado um objeto "graph", que gerenciará os elementos contidos no "paper". Após essa configuração inicial, é possível criar elementos de diversos tipos (como retângulos, círculos e setas) e, então, vinculá-los ao "graph" descrito anteriormente. Isso fará com que a biblioteca, automaticamente, desenhe esses elementos na tela e permita o usuário a interagir com eles.

Os desenvolvedores da biblioteca JointJS também disponibilizam uma forma de integrar suas funcionalidades à biblioteca React através de uma extensão denominada Rappid, mas essa extensão é um produto comercial e deve ser adquirida para poder ser usada. Por conta disso, foi necessário implementar uma integração com React por conta própria, onde apenas os elementos de interesse para o trabalho em questão foram implementados.

Com a biblioteca de diagramação JointJS configurada corretamente, foi possível começar a desenvolver um meio de modelar a Rede de Petri dentro do nosso sistema. Baseada no simulador FlexFact, a principal forma de interação com o modelo é através do mouse. O botão esquerdo do mouse é o responsável por selecionar e movimentar os elementos existentes, e o clique do botão direto abrirá um menu de contexto que apresentará as ações disponíveis para determinado elemento.

Elementos como lugares e transições são capazes de serem criados em um quadro vazio a partir do menu de contexto. Lugares são representados por círculos contendo um valor configurável, que representam seus *tokens*. Transições são representadas por barras e podem depender de eventos de entrada ou saída. As ações disponíveis ao usuário para ambos elementos são: movimentar, remover e criar um arco a partir deles.

A criação de arcos parte de um elemento do tipo lugar ou transição, e têm como destino um elemento de tipo diferente. Isto é, caso um arco parta de um lugar, esse arco precisa necessariamente ter como destino uma transição (gerando uma pré-condição), e o caso contrário também é válido (gerando uma pós-condição). Além disso, arcos também podem ser configurados para ter pesos diferentes.

Como última ferramenta para edição de uma Rede de Petri, o usuário é capaz de clicar em um elemento e será, então, redirecionado a uma tela que contém detalhes sobre aquele elemento específico. Essa tela disponibiliza, além de informações importantes, alguns campos que podem ser editados pelo usuário a fim de especificar melhor a Rede de Petri desejada. Exemplos disso são a edição de *tokens* de um lugar, o vínculo de um evento de entrada ou saída a uma transição e a edição do peso de um arco.

3.4 Configuração de eventos

Para fins de permitir a simulação de um sistema qualquer, a aplicação deverá estar ciente sobre todos os eventos que FlexFact poderá emitir ou que espera receber. Como mencionado na Seção 3.1, o protocolo escolhido para realizar a comunicação entre PetriFact e FlexFact foi o Modbus/TCP. Isso significa que, no baixo nível, a aplicação PetriFact não lidará com eventos discretos, mas sim com sinais de entrada e saída.

Felizmente, o simulador FlexFact disponibiliza uma forma de exportar a configuração Modbus/TCP de modo a facilitar a integração com os agentes externos que desejam comunicar-se com o sistema. O arquivo, com extensão ".dev", usa a sintaxe XML e contém informações tanto sobre como deverá ser realizada a conexão com o servidor (no caso, o próprio FlexFact), quanto sobre todos os eventos presentes no sistema (e quais são os sinais associados a eles).

Como visto no Capítulo 2, cada evento pode ser categorizado como de entrada ou de saída. Analogamente, esses tipos de eventos são representados de maneiras diferentes no arquivo de configuração disponibilizado pelo simulador FlexFact. Deste ponto em diante, é necessário ter em mente que dois sistemas estarão sendo mencionados concorrentemente: o modelo presente no FlexFact (simulação de um sistema fabril), e o modelo presente no PetriFact (uma Rede de Petri).

Os eventos de entrada (em relação ao FlexFact) terão sempre associados a eles uma lista de ações que deverão ser realizadas para que os requisitos de um evento sejam satisfeitos. As ações possíveis são limitadas a escrever sobre um determinado sinal com o valor 1 ("Set") ou com o valor 0 ("Clr"). Tomando como exemplo os eventos de entrada do *Conveyor Belt* descritos na Tabela 2.4, as ações descritas na coluna "Ações necessárias" seriam as mesmas descritas pelo arquivo de configuração ".dev", mas no formato especificado neste parágrafo: "ativar" seria substituído por "Set", e "desativar" por "Clr".

A configuração dos eventos de saída (também em relação ao FlexFact) seguem uma lógica semelhante. Cada evento de saída terá descrito um gatilho, que poderá ser relacionado à variação de subida ("PositiveEdge") ou de descida ("NegativeEdge") de um determinado sinal de saída. Novamente, tome como exemplo o componente *Conveyor Belt*, considerando a Tabela 2.5 que descreve seus eventos de saída. As variações descritas na coluna "Disparado por" também seriam traduzidas para o formato acima, onde "variou de 0 para 1" seria substituído por "PositiveEdge", e "variou de 1 para 0" por "NegativeEdge".

O carregamento desse arquivo de configuração para dentro da aplicação PetriFact ocorre, primeiramente, pela interface gráfica. Após a seleção do arquivo a ser carregado, o processo de renderização enviará o caminho do arquivo para o processo principal que será o responsável por ler, validar e armazenar estes dados em memória, de modo que possam ser usados para conversar diretamente com o FlexFact mais tarde.

Concluída a importação do arquivo, o processo principal retornará ao processo de renderização uma mensagem informando a lista com os eventos que foram extraídos da configuração. Esses eventos serão salvos no estado global da aplicação e poderão ser referenciados pelos elementos da Rede de Petri.

3.5 Execução do sistema

O funcionamento da aplicação PetriFact pode ser dividido em duas etapas: a primeira é a etapa de edição, que permite ao usuário criar e modificar uma Rede de Petri conforme desejar; a segunda é a etapa de execução, que desabilita qualquer interação com a rede e preocupa-se apenas em simular seu comportamento e em comunicar-se com o simulador FlexFact.

A definição original de Redes de Petri não leva em conta a existência de eventos externos à Rede de Petri. Por conta disso, não fica claro como devem ser interpretados os eventos de entrada, e nem como (e quando) devem ser emitidos eventos de saída. Como abordagem para resolver esse problema, foi implementado pelo autor deste trabalho um subconjunto do conceito de Redes de Lugar/Transição de Entrada/Saída (em inglês, *Input/Output Place/Transition Net*), também chamadas de Redes IOPT, definidas por Pais, Barros e Gomes (2005). Essa abordagem é uma extensão à definição original de Redes de Petri, apresentada na Seção 2.3, que conta com a adição de conjuntos para sinais de entrada, eventos de entrada, sinais de saída e eventos de saída.

A definição de Redes IOPT prevê que o sistema que atuará como controlador deva ter acesso à leitura de sinais de entrada, e à escrita de sinais de saída de todos os sistemas a serem controlados. Essa é uma configuração muito similar ao do presente trabalho, onde podemos considerar como controladora a aplicação PetriFact, e como controlada a aplicação FlexFact.

Resumidamente, o funcionamento de uma Rede IOPT, em relação ao controlador, é dado como segue:

1. Sinais de entrada são lidos periodicamente pelo controlador;
2. O controlador interpreta os sinais lidos no item 1 e os converte para eventos de entrada;
3. Transições ativas podem ser disparadas a partir de uma expressão booleana em função dos sinais de entrada, lidos no item 1;
4. Transições ativas podem ser disparadas a partir dos eventos de entrada, interpretados no item 2;
5. Após cada etapa de execução, o controlador pode gerar eventos de saída a partir das transições que foram disparadas;
6. Após cada etapa de execução, o controlador pode gerar sinais de saída a partir de uma expressão booleana em função da quantidade de tokens presentes em determinados lugares;
7. O controlador converte os eventos de saída em sinais de saída;
8. O controlador emite os sinais de saída resultantes dos itens 6 e 7 para o sistema sendo controlado.

Como a aplicação PetriFact está interessada apenas em reagir a eventos discretos de entrada e saída, não foi necessário incluir o mesmo tipo de tratamento para sinais. Com isso, foi possível excluir os itens 3 e 6 mencionados na lista anterior, resultando no fato de que transições serão capazes apenas de receber e emitir eventos para o FlexFact.

Com isso em mente, é possível agora explicar o funcionamento do modo de execução da aplicação PetriFact. Tudo começa com o processo de renderização, que é o responsável por permitir ao usuário modelar a Rede de Petri e configurar os eventos a serem consumidos e disparados por cada transição. Assim que a execução do modelo é solicitada, o processo de renderização desativa as interações com o usuário e envia uma cópia da Rede de Petri e do mapeamento dos eventos para o processo principal.

O processo principal, por sua vez, transforma a Rede de Petri em sua forma matricial, conforme abordado na Seção 2.3.4. A forma matricial é então armazenada em memória para uso futuro. Com isso, o processo principal está pronto para abrir a conexão com o FlexFact, que também deve estar em modo de execução e esperando uma conexão Modbus/TCP, e iniciar a execução do sistema.

Após conectado, o processo principal será responsável por ler os sinais de saída do FlexFact e transformá-los em eventos de entrada para a rede IOPT modelada pelo usuário. Baseado no estado atual do sistema (marcações e pré-condições), o processo principal de-

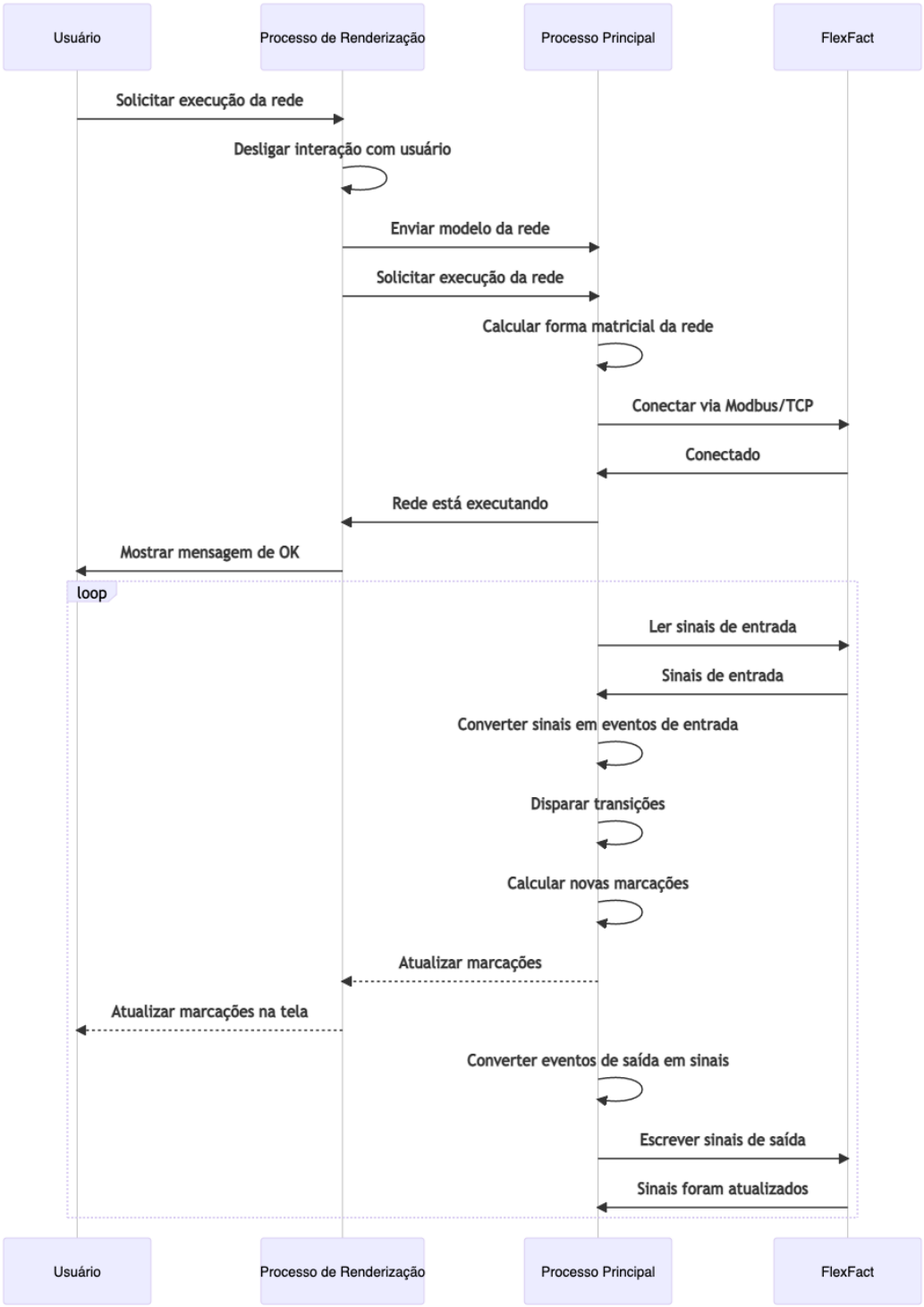
cide quais transições serão elegíveis para serem disparadas. Os eventos de entrada, então, atuarão como gatilhos para disparo dessas transições. A partir das transições disparadas, o cálculo das novas marcações é feito da mesma forma apresentada na Seção 2.3.4.

Após o cálculo das novas marcações são gerados os eventos de saída. A partir das transições que foram disparadas, o sistema decide quais eventos de saída devem ser emitidos e, então, converte-os em sinais de saída. Estes sinais de saída serão enviados para a aplicação FlexFact que, então, será capaz de interpretá-los e executar os comportamentos desejados no modelo. Ao mesmo tempo, o processo de renderização é informado sobre a atualização das marcações e faz as devidas atualizações na tela.

Esse processo de leitura, execução e escrita acontece periodicamente, de maneira síncrona, múltiplas vezes por segundo, finalizando apenas em dois casos: (1) a aplicação PetriFact perdeu a conexão com o simulador FlexFact, ou (2) o usuário solicitou à aplicação PetriFact para que parasse de executar. Quando isso acontece, então, a aplicação fecha a conexão aberta e restaura as marcações para os valores originais.

A Figura 3.2 apresenta um diagrama de sequência que representa o *loop* principal da aplicação, sem considerar a parada e desconexão entre as aplicações.

Figura 3.2 – Funcionamento do modo de execução do PetriFact



Fonte: Autor

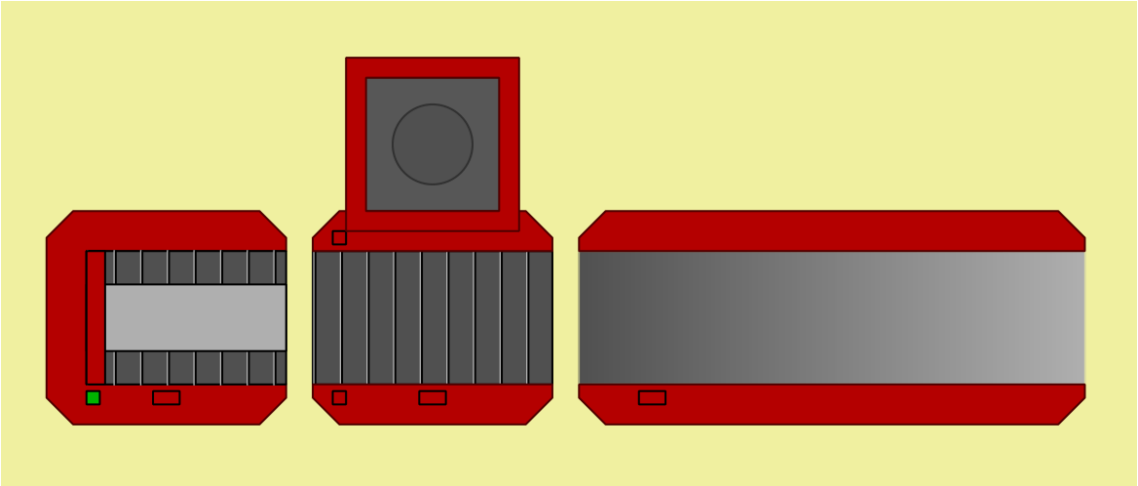
4 RESULTADOS

Neste capítulo, primeiramente, será demonstrado o funcionamento da ferramenta PetriFact através de um exemplo. Com base nisso, serão discutidos quais objetivos foram alcançados, de acordo com o que foi especificado no Capítulo 1.

4.1 Funcionamento da ferramenta

A explicação do funcionamento da ferramenta ocorrerá através de um exemplo prático. Antes de tudo, será necessário ter em mãos um sistema modelado usando o simulador FlexFact. A fim de tornar esse exemplo palpável para o leitor, será utilizado um sistema simples, representado pela Figura 4.1.

Figura 4.1 – Exemplo de sistema modelado no FlexFact



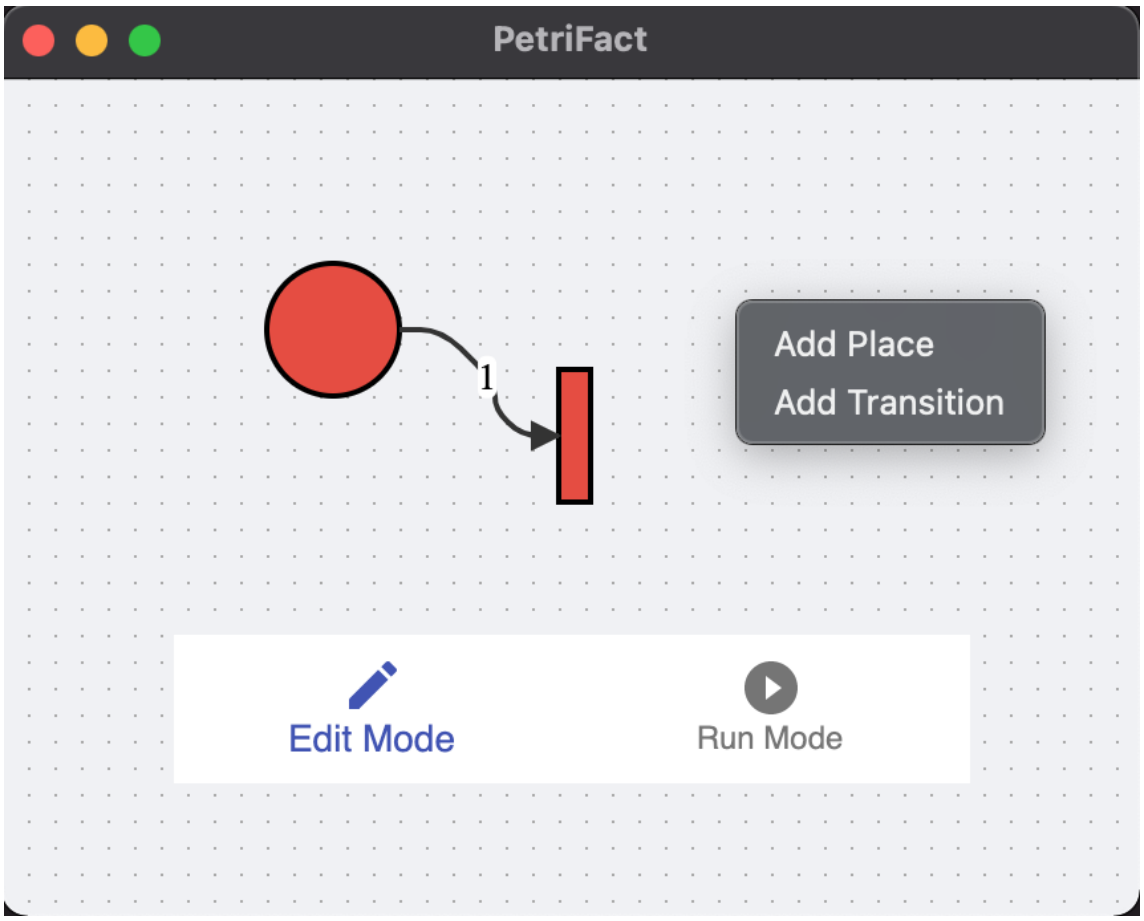
Fonte: Autor

Esse sistema contém uma unidade alimentadora, uma máquina processadora e um escorregador de saída. O objetivo desse sistema é de, ao receber um pacote, transportá-lo até a máquina processadora, que pausará a esteira, realizará uma operação sobre o pacote e, então, retomará o andamento da esteira. Com isso, o pacote será então descartado no escorregador de saída.

Após modelado o sistema no simulador FlexFact, o usuário precisará exportar a configuração Modbus/TCP do modelo, disponível através de sua interface gráfica. Os eventos contidos nessa configuração podem ser visualizados na Tabela 4.1. Concluída esta etapa, o usuário poderá executar a aplicação PetriFact localmente em seu computador.

A primeira tela apresentada pela aplicação será de boas-vindas, oferecendo opções para criação ou carregamento de um projeto. Para este exemplo será criado um novo. Logo após isso, a aplicação solicitará o carregamento do arquivo de configuração Modbus/TCP, gerado anteriormente, que fará com que PetriFact seja capaz de preparar o projeto e, então, apresentar a tela principal ao usuário, apresentada na Figura 4.2.

Figura 4.2 – Tela principal da aplicação PetriFact



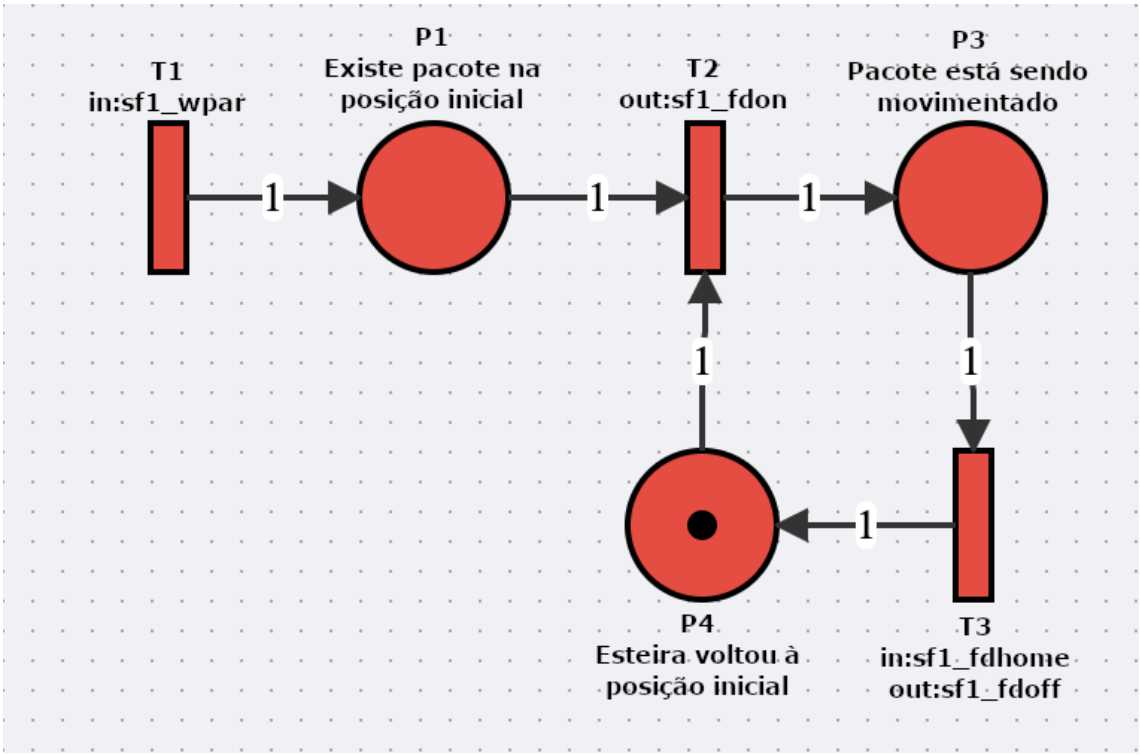
Fonte: Autor

No início, a tela principal consiste de um "quadro em branco" e de dois botões no canto inferior da tela: "Edit Mode", usado para habilitar a edição da Rede de Petri (padrão), e "Run Mode", usado para transformar a rede modelada no PetriFact em um controlador para o sistema modelado no FlexFact. Novos elementos podem ser criados a partir do menu de contexto, aberto através do clique do botão direito do mouse no quadro. Elementos existentes podem ser arrastados pela tela com o mouse e, também, podem ser manipulados através de cliques. Um clique do botão esquerdo abre uma janela com detalhes sobre o elemento em questão, permitindo ao usuário alterar seus atributos (como o número de *tokens* de um lugar, por exemplo). Já um clique do botão direito abrirá

o menu de contexto, permitindo ao usuário realizar ações sobre o elemento em questão (como, por exemplo, removê-lo da tela).

O próximo passo é, então, modelar uma Rede de Petri que seja capaz de executar o exemplo como esperado. Aproveitando-se do fato de que a aplicação PetriFact permite gerar múltiplas Redes de Petri separadamente, é possível modelar o comportamento de cada componente isoladamente. Com o apoio da Tabela 4.1, foi possível gerar as redes apresentadas nas Figuras 4.3 e 4.4.

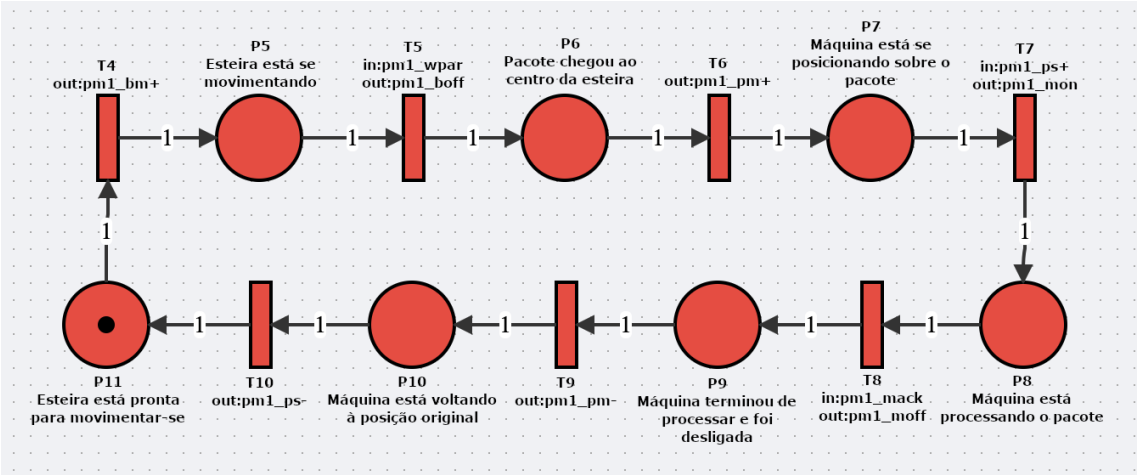
Figura 4.3 – Rede de Petri relativa à unidade alimentadora



Fonte: Autor

Com o modelo pronto e com seus eventos mapeados corretamente é, então, possível executar a simulação. Primeiramente, é necessário ativar a comunicação via Modbus/TCP no simulador FlexFact e então inicializar a simulação. A última etapa é, agora, habilitar o modo de execução na aplicação PetriFact clicando no botão "Run Mode". Isso fará com que ambas as ferramentas comecem a conversar entre si, e que a simulação possa ser visualizada em qualquer uma das aplicações: no FlexFact, na forma de um sistema flexível de manufatura, no PetriFact, na forma de uma Rede de Petri.

Figura 4.4 – Rede de Petri relativa à máquina processadora



Fonte: Autor

4.2 Objetivos alcançados

A ferramenta desenvolvida apresenta uma interface gráfica que possibilita a modelagem de Redes de Petri de maneira simples, usando apenas o mouse. Ela também é capaz de comunicar-se de maneira funcional com o simulador FlexFact, trocando sinais múltiplas vezes por segundo através do protocolo Modbus/TCP com um intervalo aproximado de 100 milissegundos entre cada iteração.

Além disso, a ferramenta traduz os sinais de entrada e saída em eventos, que podem, então, ser utilizados pela Rede de Petri como gatilhos para as transições configuradas. Finalmente, utilizando a definição matemática em conjunto com a forma matricial, a ferramenta consegue calcular o próximo estado da rede e emitir eventos de volta para o agente externo, FlexFact.

Somando isso à explicação do funcionamento da aplicação, apresentada na seção anterior, é possível inferir que a aplicação, de fato, satisfez o objetivo geral deste trabalho: desenvolver uma aplicação de síntese e simulação de Redes de Petri capaz de controlar um sistema sendo simulado pela ferramenta FlexFact (e, potencialmente, outros sistemas que respeitem o mesmo contrato Modbus/TCP).

Tabela 4.1 – Eventos de entrada e saída do exemplo (em relação ao FlexFact)

Evento	*Tipo*	*Descrição*
pm1_bm+	Entrada	Mover esteira da máquina para leste
pm1_bm-	Entrada	Mover esteira da máquina para oeste
pm1_boff	Entrada	Parar esteira da máquina
pm1_moff	Entrada	Desligar máquina
pm1_mon	Entrada	Ligar máquina
pm1_pm+	Entrada	Mover máquina em direção sul
pm1_pm-	Entrada	Mover máquina em direção norte
pm1_poff	Entrada	Parar de mover máquina
sf1_fdoff	Entrada	Parar esteira da unidade alimentadora
sf1_fdon	Entrada	Mover esteira da unidade alimentadora
pm1_mack	Saída	Máquina terminou de processar
pm1_mrqu	Saída	Máquina pronta para processar
pm1_ps+	Saída	Máquina chegou ao destino sul
pm1_ps-	Saída	Máquina chegou ao destino norte
pm1_wpar	Saída	Pacote chegou até máquina
pm1_wplv	Saída	Pacote deixou a máquina
sf1_fdhome	Saída	Esteira está na posição inicial
sf1_wpar	Saída	Pacote chegou na unidade alimentadora
sf1_wplv	Saída	Pacote deixou a unidade alimentadora
xs1_wpar	Saída	Pacote chegou à saída
xs1_wplv	Saída	Pacote deixou a saída

Fonte: Autor

5 CONCLUSÃO

O desenvolvimento do presente trabalho resultou na implementação de uma ferramenta, denominada PetriFact, que visa demonstrar a aplicação de Redes de Petri à sistemas a eventos discretos. De modo a possibilitar a simulação de um sistema desse tipo, foi utilizado o software FlexFact como apoio para construí-lo, que provê um ambiente virtual onde é possível modelar sistemas flexíveis de manufatura e simular seu comportamento.

As principais dificuldades encontradas durante a implementação foram a escolha das tecnologias a serem utilizadas, necessitando reavaliação frequente a fim de garantir que estariam atendendo a todos os requisitos propostos, e o desenvolvimento da interface gráfica como um todo, que mostrou-se como sendo muito trabalhoso.

Como trabalhos futuros, é possível citar a resolução de conflitos ao disparar transições. No momento da escrita desta monografia, PetriFact não está preparado para lidar com casos onde duas transições ativas distintas, que dependam das mesmas pré-condições, sejam disparadas. Caso ocorram, conflitos podem resultar em *tokens* sendo distribuídos errôneamente. Outro ponto que pode ser explorado é a adição de suporte ao protocolo Simplenet ou outro tipo de comunicação baseada em eventos, que potencialmente otimizaria o uso de recursos de rede (evitando múltiplas chamadas por segundo), oferecendo maior oportunidade de escalabilidade à ferramenta.

Como nota final, o autor pretende manter o projeto como código-aberto na plataforma GitHub, possibilitando a qualquer pessoa visualizar o código-fonte e contribuir com o desenvolvimento do mesmo. O projeto pode ser encontrado em https://github.com/gabrieltjunior/petrifact.

REFERÊNCIAS

ABRAMOV, Dan. **React Redux**. 2015. Disponível em: https://react-redux.js.org/. Acesso em: 11 mai. 2021.

BIERMAN, Gavin; ABADI, Martín; TORGERSEN, Mads. Understanding TypeScript. *In* JONES, Richard (Ed.). **ECOOP 2014 – Object-Oriented Programming**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. p. 257–281. ISBN 978-3-662-44202-9.

CASSANDRAS, Christos G.; LAFORTUNE, Stephane. **Introduction to Discrete Event Systems**. 2nd. [*S. l.*]: Springer Publishing Company, Incorporated, 2008. ISBN 978-0-387-68612-7.

CLIENT.IO. **JointJS**. 2013. Disponível em: https://www.jointjs.com/. Acesso em: 11 mai. 2021.

CURY, José Eduardo Ribeiro. Teoria de Controle Supervisório de Sistemas a Eventos Discretos. *In*. SIMPÓSIO Brasileiro de Automação Inteligente, V. [*S. l.: s. n.*], 2001. Disponível em: http://www.pb.utfpr.edu.br/mt/pdfs/Doutorado/ControleSuperv/apostila.pdf. Acesso em: 7 mai. 2021.

ELECTRON REACT BOILERPLATE. **Electron React Boilerplate**. 2015. Disponível em: https://github.com/electron-react-boilerplate/electron-react-boilerplate. Acesso em: 11 mai. 2021.

ELOUAFI, Yassine. **Redux-Saga**. 2015. Disponível em: https://redux-saga.js.org/. Acesso em: 11 mai. 2021.

FACEBOOK OPEN SOURCE. **React**. 2013. Disponível em: https://reactjs.org/. Acesso em: 11 mai. 2021.

FGDES. **DESTool**. 2008. Disponível em: https://fgdes.tf.fau.de/destool/. Acesso em: 6 mai. 2021.

FGDES. **FlexFact**. 2011. Disponível em: https://fgdes.tf.fau.de/flexfact.html. Acesso em: 6 mai. 2021.

GOOGLE. **Chromium**. 2008. Disponível em: https://www.chromium.org/. Acesso em: 11 mai. 2021.

MODBUS ORGANIZATION, Inc. **Modbus Application Protocol Specification**. 2012. Disponível em: https://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf. Acesso em: 6 mai. 2021.

OPENJS FOUNDATION. **Electron**. 2013. Disponível em: https://www.electronjs.org/. Acesso em: 11 mai. 2021.

OPENJS FOUNDATION. **NodeJS**. 2009. Disponível em: https://nodejs.org/en/. Acesso em: 11 mai. 2021.

PAIS, R.; BARROS, S.P.; GOMES, L. A tool for tailored code generation from Petri net models. *In*. 2005 IEEE Conference on Emerging Technologies and Factory Automation. [*S. l.: s. n.*], 2005. v. 1, 8 pp.–864. DOI: 10.1109/ETFA.2005.1612615.

PETERSON, James L. Elementary Net Systems. *In*. Edição: Wolfgang Reisig e Grzegorz Rozenberg. [*S. l.*]: Prentice Hall, 1981.

PETERSON, James L. Petri Nets. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 9, n. 3, p. 223–252, 1977.

ZURAWSKI, Richard; ZHOU, Mengchu. Petri net and industrial application: A tutorial. **Industrial Electronics, IEEE Transactions on**, v. 41, p. 567–583, jan. 1995. DOI: 10.1109/41.334574.