Bacharelado em Sistemas de Informação
Gerência de Processos de Negócio
Visualizando Modelos BPMN e Redes de Petri Equivalentes

Mateus Conrad B. da Costa

Instituto Federal do Espírito Santo - Campus Serra

December 9, 2024

O que é Process Mining?

- Process Mining é uma área que combina mineração de dados com a análise de processos de negócio.
- Permite descobrir, monitorar e melhorar processos reais com base em dados de execução de sistemas de informação.
- O **pm4py** é uma biblioteca Python que fornece ferramentas para realizar mineração de processos.

Biblioteca pm4py

- **pm4py** (Process Mining for Python) é uma biblioteca open source para análise de processos de negócio.
- Facilita a importação de logs de eventos, descoberta de modelos, verificação de conformidade, análise de desempenho e visualização de processos.
- Nesta introdução, utilizaremos **pm4py** para visualizar e converter um modelo BPMN em uma Rede de Petri.

Passo 1: Seleção do Arquivo BPMN

- Utilizamos um script em Python para permitir ao usuário selecionar um arquivo BPMN.
- A função `selecionar_arquivo_bpmn()` faz uso da biblioteca Tkinter para abrir uma caixa de diálogo e permitir a seleção do arquivo.
- Isso torna o processo mais interativo para os alunos.

Código: Seleção do Arquivo BPMN

```
import tkinter as tk
from tkinter import filedialog

def selecionar_arquivo_bpmn():
    root = tk.Tk()
    # Esconde a janela principal do Tkinter
    root.withdraw()
    file_path = filedialog.askopenfilename(
        initialdir=".",
        title="Selecione o arquivo BPMN",
        filetypes=[("Arquivos BPMN", "*.bpmn")]
# Apenas arquivos .bpmn
    )
    return file_path
```

Passo 2: Visualização do Modelo BPMN

- Após selecionar o arquivo BPMN, o modelo é carregado utilizando o `bpmn_importer` da **pm4py**.
- A visualização do modelo é feita por meio da função `visualizer.apply()`, que gera uma imagem do diagrama BPMN.

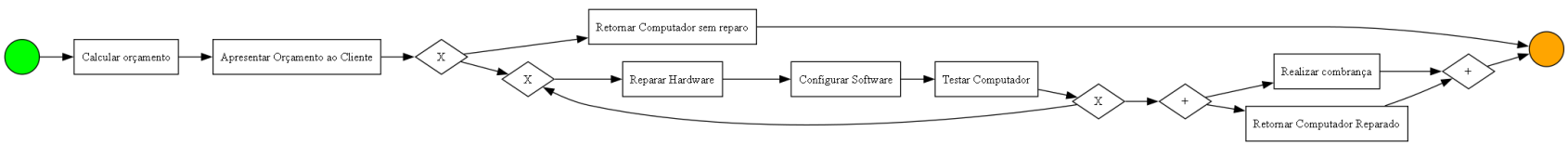

Figure: Visualização do Modelo BPMN - Reparo de Computador

Código: Visualização do Modelo BPMN

```
import pm4py
from pm4py.visualization.bpmn import visualizer

def visualize_bpmn(file_path):
    # Carregar o modelo BPMN a partir do arquivo
    bpmn_model = pm4py.read_bpmn(file_path)
    # Gerar a visualiza  o do modelo BPMN
    gviz = visualizer.apply(bpmn_model)

    #  visualizar no ambiente gr fico
    visualizer.view(gviz)

    # For ar o fechamento da visualiza  o
    input("Pressione Enter para
----fechar a visualiza  o...")
```

O que é uma Rede de Petri?

- Redes de Petri são uma ferramenta gráfica e matemática utilizada para modelar sistemas distribuídos e concorrentes.
- São compostas por:
  - **Lugares** (círculos) - representam estados do sistema.
  - **Transições** (retângulos) - representam eventos que mudam o estado.
  - **Arcos** - conectam lugares e transições, indicando a direção da mudança de estado.
- Um elemento importante das redes de Petri são os **tokens**, que representam a presença de um recurso ou estado em um lugar.
- As transições são disparadas quando todos os lugares de entrada contêm tokens, e ao serem disparadas, consomem tokens dos lugares de entrada e adicionam tokens nos lugares de saída.

Exemplos de Redes de Petri

- Abaixo estão exemplos de redes de Petri.

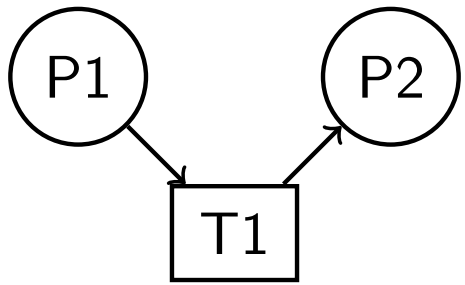

Figure: Exemplo de Rede de Petri Simples

Gateways em Redes de Petri

- Em modelos de processos, podemos ter diferentes tipos de gateways para controlar o fluxo.
- Vamos ver como representar gateways XOR e AND em uma Rede de Petri.

Gateway XOR em Redes de Petri

- Um **gateway XOR** em uma Rede de Petri pode ser representado por uma transição que consome tokens de um lugar e os distribui para diferentes lugares, mas apenas uma das saídas é habilitada a cada vez.

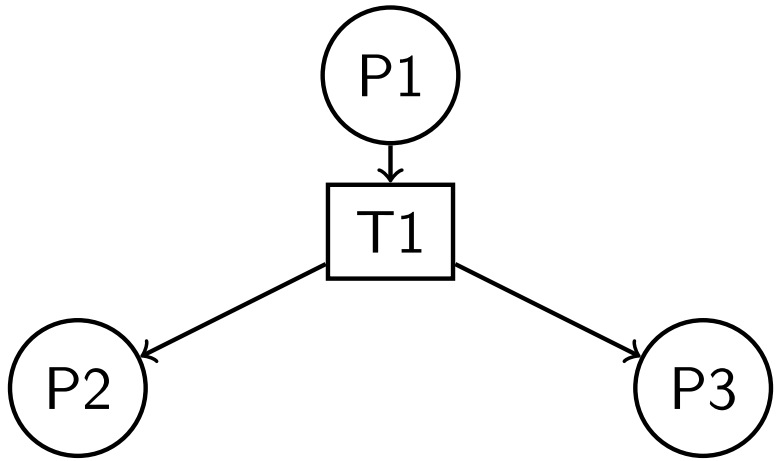

Figure: Gateway XOR Representado em Rede de Petri

Gateway AND Fork em Redes de Petri

- Um **gateway AND Fork** em uma Rede de Petri é utilizado para iniciar o paralelismo, distribuindo tokens de um lugar para vários lugares simultaneamente.
- Representa um ponto em que o fluxo do processo se divide em múltiplos fluxos paralelos.

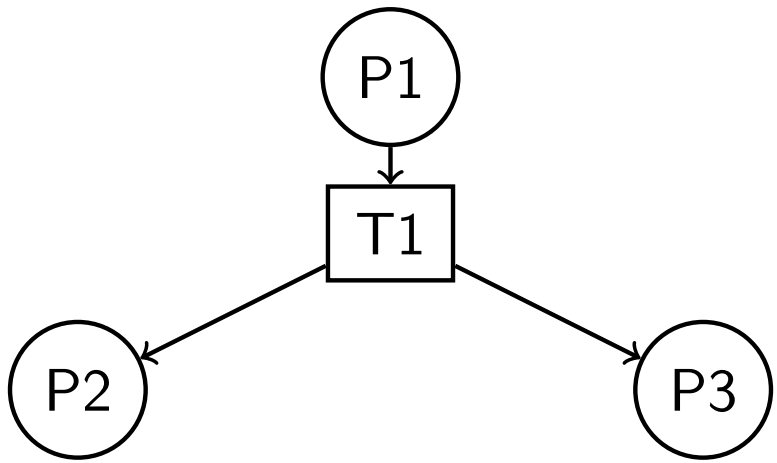

Figure: Gateway AND Fork Representado em Rede de Petri

Gateway AND em Redes de Petri

- Um **gateway AND** em uma Rede de Petri pode ser representado por uma transição que consome tokens de múltiplos lugares e os distribui para múltiplos lugares de saída simultaneamente.

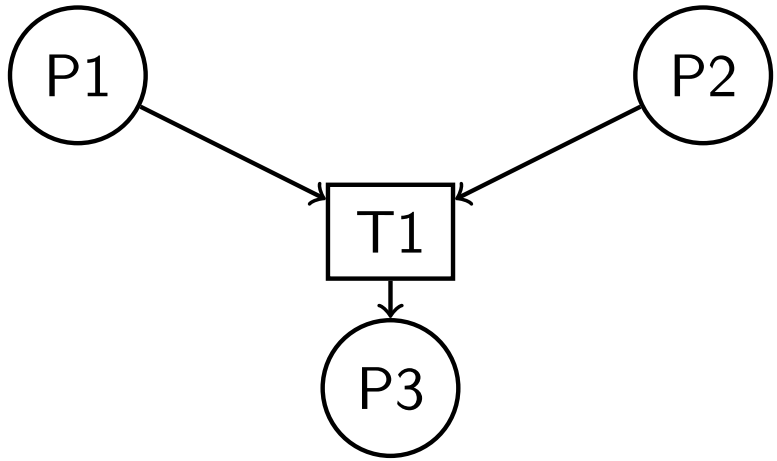

Figure: Gateway AND Representado em Rede de Petri

Passo 3: Converter BPMN para Rede de Petri

- Utilizamos a função `bpmn_converter` da **pm4py** para converter o modelo BPMN em uma Rede de Petri equivalente.
- Isso facilita a análise da lógica do processo e permite simular o comportamento do sistema.

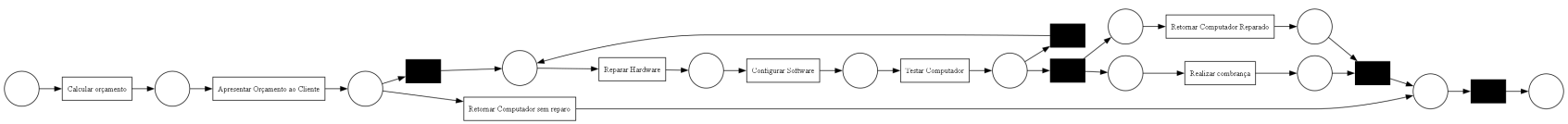

Figure: Rede de Petri - Reparo de Computador

Código: Conversão para Rede de Petri

```
from pm4py.objects.conversion.bpmn import
converter as bpmn_converter

def convert_bpmn_to_petri(file_path):
    # Carregar o modelo BPMN
    bpmn_graph = pm4py.read_bpmn(file_path)
    # Converter para Rede de Petri
    net, im, fm = bpmn_converter.apply(bpmn_graph,
    variant=bpmn_converter.Variants.TO_PETRI_NET)
    return net, im, fm
```

Passo 4: Visualização da Rede de Petri

- A Rede de Petri resultante é visualizada utilizando o `pn_visualizer` da **pm4py**.
- Isso ajuda a compreender como os estados e transições se relacionam no modelo de processo.

Código: Visualização da Rede de Petri

```
from pm4py.visualization.petri_net import
visualizer as pn_visualizer

def visualize_petri_net(net, im, fm):
    # Gerar o modelo visual de Rede de Petri
    gviz = pn_visualizer.apply(net, im, fm)
    # Tentar visualizar no ambiente
    pn_visualizer.view(gviz)
```

Google Colab

- O **Google Colab** é uma plataforma baseada na nuvem para execução de código Python diretamente no navegador.
- Popular entre cientistas de dados, desenvolvedores e educadores pela simplicidade e acessibilidade.
- Utilizado para análises, visualizações e tarefas que exigem alto desempenho, como aprendizado de máquina.

Principais Recursos

- **Acesso à Infraestrutura de Computação:**
  Permite uso gratuito de **GPUs** e **TPUs** para tarefas de alto desempenho.
- **Integração com o Google Drive:**
  Armazene, carregue e salve arquivos diretamente no Google Drive.
- **Colaboração em Tempo Real:**
  Compartilhamento e edição colaborativa similar ao Google Docs.
- **Bibliotecas Pré-Instaladas:**
  Inclui `NumPy`, `Pandas`, `Matplotlib` e outras bibliotecas populares.

Google Colab

1. Acesse: `https://colab.research.google.com`
2. Faça login com sua conta Google.
3. Clique em **Novo Notebook** para criar seu ambiente de trabalho.
4. Conecte o ambiente clicando em **Conectar** (canto superior direito).

Carregando Bibliotecas e Arquivos

Instalar bibliotecas (se necessário):

```
!pip install pm4py
```

Carregar arquivos locais:

```
from google.colab import files
uploaded = files.upload()
```

Conectar ao Google Drive:

```
from google.colab import drive
drive.mount('/content/drive')
```

Exemplo de Código no Colab

```
# Instalar a biblioteca pm4py
!pip install pm4py

# Importar bibliotecas
import pm4py

# Criar um log de eventos fictício
from pm4py.objects.log.importer.xes import
factory as xes_importer

# Carregar um log de eventos de exemplo
log = pm4py.read_xes('seu_arquivo.xes')

# Exibir informações básicas do log
pm4py.view_log(log)
```

Passo 5: Simulação do Processo e Geração de Logs

- Utilizamos a função `simulator` para gerar logs de eventos a partir da Rede de Petri.
- Esses logs simulam a execução do processo e fornecem dados sobre os tempos de execução e duração de atividades.
- O log gerado pode ser exportado para um arquivo `.xes` para análise posterior.

Função: Geração de Logs

Descrição:

- A função `generate_logs` gera logs sintéticos baseados em uma rede de Petri fornecida.
- Exporta os logs gerados em formato XES para análise posterior.

Função: Geração de Logs

```
def generate_logs(petri_net, initial_marking,
                  output_file="synthetic_log.xes"):
# Gerar logs  do processo
    simulation_log = simulator.apply(
    petri_net, initial_marking,
    variant=simulator.Variants.BASIC_PLAYOUT)
    print_logs(simulation_log)

# Exportar o log de  para um arquivo XES
    xes_exporter.apply(simulation_log,
    output_file)
    print(f"Log salvo em:
----{os.path.abspath(output_file)}")
```

Função: Impressão de Logs

Descrição:

- A função `print_logs` imprime os logs gerados, separando cada caso.
- Detalha atividades e timestamps de cada evento no log.

Código:

```
def print_logs(simulation_log):
for i, case in enumerate(simulation_log):
    print(f"Caso {i + 1}:")
    for event in case:
        print(f"  Atividade: {event['concept:name']}
--------Timestamp: {event['time:timestamp']}")
        print("-" * 30)
```

Exemplo de Uso das Funções

Código:

```
# Gerar e imprimir logs
generate_logs(petri_net, initial_marking)

# Exemplo de saída impressa no console:
# Caso 1:
# Atividade: Start, Timestamp: 2024-12-01T10:00:00
# Atividade: Process, Timestamp: 2024-12-01T10:05:0
# ----------------------------------------
# Caso 2:
# Atividade: Start, Timestamp: 2024-12-01T10:01:00
# Atividade: Process, Timestamp: 2024-12-01T10:06:0
# ----------------------------------------
```

Dicas Finais

- Certifique-se de verificar a estrutura da rede de Petri antes de gerar logs.
- O arquivo XES gerado pode ser importado para ferramentas de análise de processos, como ProM ou Disco.
- Use a função `print_logs` para validar visualmente o conteúdo dos logs gerados.

Função: Análise de Processo

Descrição:

- A função `analyze_process` analisa os logs para identificar:
  - Tempos médios de execução de cada atividade.
  - A tarefa mais demorada.
  - O tempo de ciclo médio do processo.
  - A sequência com maior tempo médio.

Código:

```
def analyze_process(simulation_log):
    """
    Analisa os logs para calcular
    estatisticas do processo.
    Parametros:
    simulation_log (list): Log gerado
    pela rede de Petri.
    Retorna: dict: Estatisticas do processo
    incluindo tempos medios, tarefa mais demorada,
    tempo de ciclo da sequencia mais lenta.
```

Função: Análise de Processo

```
activity_times = defaultdict(list)
sequence_times = defaultdict(list)
# Processar cada caso no log
for case in simulation_log:
    for i, event in enumerate(case):
        if i > 0:
            prev_event = case[i - 1]
            delta = event['time:timestamp'] -
            prev_event['time:timestamp']
            sequence = (prev_event['concept:name'],
            event['concept:name'])
            sequence_times[sequence].append(delta.
            total_seconds())
            activity_times[event['concept:name']].
            append(event['time:timestamp'])
```

Função: Análise de Processo

```
# Calcular estatisticas
activity_avg = {act:
round(pd.Series(times).diff().mean().
total_seconds())
for act, times in activity_times.items()}
most_time = max(activity_avg,
key=activity_avg.get)
cycle_time_avg = round(sum([sum(times)
for times
in sequence_times.values()]) /
len(simulation_log))
longest_sequence = max(sequence_times,
key=lambda
x: pd.Series(sequence_times[x]).mean())

# Converter tempos para segundos
readable_activity_avg =
{act: f"{avg}-segundos"
```

Função: Análise de Processo

```
# Exibir os resultados
print("--- Analise do Processo ---")
print("Tempos medios por atividade:",
readable_activity_avg)
print("Tarefa mais demorada:", most_time,
f"({activity_avg[most_time]} segundos)")
print("Tempo de ciclo medio:",
readable_cycle_time_avg)
print("Sequencia com maior tempo medio:",
readable_longest_sequence)
# Retornar os resultados para uso posterior
return {
"activity_avg": readable_activity_avg,
"most_time": most_time,
"cycle_time_avg": readable_cycle_time_avg,
"longest_sequence": readable_longest_sequence
}
```

Exemplo de Uso das Funções

Código:

```
# Gerar e analisar logs
simulation_log = generate_logs(petri_net,
initial_marking)
analyze_process(simulation_log)

# Exemplo de sa da impressa no console:
# ---- An lise do Processo ----
# Tempos m dios por atividade: {'Start': 5.0,
'Process': 10.0}
# Tarefa mais demorada: Process
# Tempo de ciclo m dio: 30.0
# Sequ ncia com maior tempo m dio:
('Start', 'Process')
```