VINÍCIUS LEOBET BREGOLI

DESENVOLVIMENTO DE GERADOR DE MODELOS DE SIMULAÇÃO PARA TOMADA DE DECISÃO NO CURTO PRAZO USANDO PROCESS MINING

Curitiba

2025

VINÍCIUS LEOBET BREGOLI

DESENVOLVIMENTO DE GERADOR DE MODELOS DE SIMULAÇÃO PARA TOMADA DE DECISÃO NO CURTO PRAZO USANDO PROCESS MINING

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do grau de Bacharel em Engenharia de Computação.

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ – PUCPR

ESCOLA POLITÉCNICA

CURSO DE ENGENHARIA DE COMPUTAÇÃO

Orientador: Prof. Dr. Edson Emílio Scalabrin

Curitiba

2025

Resumo

A tomada de decisão no curto prazo em ambientes complexos, como centros cirúrgicos, exige ferramentas capazes de integrar dados históricos e informações em tempo real. Nesse contexto, a mineração de processos (Process Mining – PM) e a simulação computacional emergem como tecnologias para apoiar gestores na alocação eficiente de recursos, detecção de desvios e previsão de cenários. Este trabalho propõe o desenvolvimento de um gerador de modelos de simulação baseado em PM, com foco em reduzir o esforço humano e o tempo necessário para a construção de modelos. O gerador utiliza logs de eventos no formato XES como fonte de dados para criar automaticamente modelos de simulação, permitindo avaliar alternativas de curto prazo e apoiar a tomada de decisão operacional. Como resultado, foi desenvolvido um protótipo funcional que integra cinco funcionalidades principais: análise automática de logs, mineração de processos utilizando o algoritmo Inductive Miner, simulação de eventos com geração de logs sintéticos, validação comparativa entre logs originais e simulados, e cálculo de indicadores ORE (Operating Room Effectiveness). O protótipo foi implementado como uma interface web interativa utilizando Streamlit, e validado com dados reais de agendamentos cirúrgicos de um centro cirúrgico. A ferramenta permite aos gestores extrair automaticamente modelos de processo (redes de Petri e árvores de processo), gerar cenários simulados para análise de capacidade e desempenho, e calcular métricas de efetividade operacional, contribuindo para maior eficiência, redução de custos e melhor qualidade no atendimento.

Palavras-chave: Mineração de Processos, Simulação Computacional, Tomada de Decisão, Otimização, Agendamento.

Abstract

Short-term decision-making in complex environments, such as surgical centers, requires tools capable of integrating historical data and real-time information. In this context, Process Mining (PM) and computer simulation emerge as key technologies to support managers in efficient resource allocation, deviation detection, and scenario prediction. This work proposes the development of a simulation model generator based on PM, focusing on reducing human effort and the time required to build models. The generator uses event logs in XES format as a data source to automatically create simulation models, allowing the evaluation of short-term alternatives and supporting operational decision-making. As a result, a functional prototype was developed integrating five main functionalities: automatic log analysis, process mining using the Inductive Miner algorithm, event simulation with synthetic log generation, comparative validation between original and simulated logs, and calculation of ORE (Operating Room Effectiveness) indicators. The prototype was implemented as an interactive web interface using Streamlit, and validated with real surgical scheduling data from a surgical center. The tool enables managers to automatically extract process models (Petri nets and process trees), generate simulated scenarios for capacity and performance analysis, and calculate operational effectiveness metrics, contributing to greater efficiency, cost reduction, and improved service quality.

Keywords: Process Mining, Computer Simulation, Decision-Making, Optimization, Scheduling.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Organizações modernas enfrentam o desafio de transformar o crescente volume de dados operacionais em decisões rápidas e baseadas em evidências. Enquanto sistemas informatizados geram logs detalhados de execução de processos, a análise desses dados permanece majoritariamente manual e dependente de expertise humana. A *mineração de processos* (PM) permite extrair modelos descritivos de processos reais, mas oferece primariamente uma visão retrospectiva. A *simulação de eventos discretos* (DES), por outro lado, viabiliza a análise preditiva de cenários, porém sua construção manual é intensiva e inviável para decisões de curto prazo.

O presente trabalho desenvolveu um **gerador automatizado de modelos de simulação baseado em mineração de processos**, capaz de transformar logs de eventos estruturados em modelos DES parametrizados sem intervenção manual. O protótipo foi validado com dados reais do setor hospitalar e projetado para ser generalizável a diferentes domínios organizacionais que disponham de registros no padrão XES. Essa integração preenche lacuna identificada na literatura: a necessidade de automação end-to-end entre extração de conhecimento (PM) e predição operacional (DES).

1.1 Contexto e Problema

A crescente complexidade dos ambientes organizacionais exige capacidade contínua de adaptação e resposta rápida. Setores como saúde, manufatura e logística compartilham desafios recorrentes: alocação eficiente de recursos, detecção de gargalos e melhoria contínua de desempenho. Em todos esses contextos, decisões de curto prazo — aquelas que precisam ser tomadas em intervalos de horas ou dias — exercem influência direta sobre a produtividade e eficiência operacional.

Com a intensificação da digitalização, grandes volumes de dados passaram a ser gerados em tempo real. Contudo, a transformação desses dados em conhecimento útil ainda depende, em grande medida, da experiência humana e de análises manuais, sujeitas a vieses cognitivos e limitações de tempo de resposta.

A *mineração de processos* (PM) consolida-se como abordagem para extração de conhecimento estruturado a partir de logs de eventos, com aplicações demonstradas em domínios complexos. Entretanto, PM oferece predominantemente uma visão descritiva e diagnóstica. A simulação de eventos discretos, por outro lado, permite avaliar cenários alternativos, mas sua construção manual é intensiva e inviável para decisões de curto prazo que exigem reação rápida a eventos inesperados.

O desafio central, portanto, é: **como automatizar a geração de modelos de simulação**

baseados em dados reais, de modo a apoiar decisões operacionais de curto prazo? Trabalhos como PM4SOS (FERRONATO, 2022) demonstraram potencial dessa integração em contextos específicos (hospitalar), mas ainda com dependência de intervenções manuais e limitada generalização. A lacuna identificada reside na necessidade de uma solução automatizada, end-to-end e generalizável, que reduza o esforço de modelagem e amplie a aplicabilidade da técnica ao suporte à decisão operacional.

1.2 Delimitação do Escopo

Escopo técnico: Este trabalho aborda processos discretos com eventos estruturados no padrão XES. A solução integra descoberta de processos (Inductive Miner via PM4PY), extração estatística de distribuições de tempos, identificação automática de recursos, e geração de modelos parametrizados em SimPy.

Escopo temporal: O foco é apoiar decisões operacionais de curto prazo, definidas como aquelas que necessitam resposta em horizonte de horas a dias (não inclui planejamento estratégico de longo prazo ou otimização em tempo real).

Escopo de aplicação: Aplicável a processos organizacionais que possuam logs de eventos estruturados conforme padrão XES, incluindo contextos hospitalares, administrativos, logísticos e de manufatura.

Limitações explícitas: O trabalho não aborda: (i) processos contínuos ou híbridos (contínuo-discreto), (ii) tratamento de sensores de tempo real não estruturados, (iii) otimização multicritério automática (focos apenas em simulação), (iv) modelos com sincronização complexa entre múltiplos recursos, (v) tratamento de incerteza estocástica além das distribuições estatísticas estimadas.

1.3 Motivação

Organizações modernas enfrentam uma urgência operacional: decisões precisam ser tomadas em horas ou dias, não em semanas. A maioria dos dados operacionais coletados continua subutilizada, pois análises descritivas retrospectivas não oferecem suporte efetivo a decisões ágeis. O framework PM4SOS (FERRONATO, 2022) demonstrou que a integração entre mineração de processos e simulação é viável em contextos hospitalares, porém permanece restrita e com etapas manuais.

A motivação deste trabalho é clara: criar uma solução que (i) elimine intervenções manuais na geração de modelos, (ii) reduza o tempo de análise de semanas para minutos, (iii) seja aplicável a diferentes domínios sem reprogramação, e (iv) democratize o acesso a ferramentas de simulação para analistas e gestores sem expertise em modelagem formal. Acredita-se que

automatização completa do pipeline PM-DES pode transformar simulação de instrumento ocasional de especialistas em ferramenta cotidiana de apoio à decisão operacional.

1.4 Soluções Similares

Diversas pesquisas têm buscado integrar *Process Mining* (PM) e Simulação de Eventos Discretos como forma de aprimorar a compreensão e a predição do comportamento dos processos reais. Entretanto, a maioria das soluções existentes mantém um alto grau de dependência de intervenção manual, especialmente nas etapas de modelagem, parametrização e calibração.

Entre as iniciativas mais influentes, destaca-se a metodologia proposta por (MARUŞTER; BEEST, 2009), que combina mineração de processos e simulação para o redesenho organizacional. O método parte de logs reais para gerar modelos *As-Is*, simulá-los e compará-los com versões otimizadas *To-Be*, permitindo estimar ganhos de desempenho. Apesar de pioneiro, o processo de conversão dos modelos minerados em modelos simuláveis requer ajustes manuais e conhecimento técnico em modelagem formal (como redes de Petri coloridas).

No contexto hospitalar, o framework PM4SOS, desenvolvido por (FERRONATO, 2022), integra mineração de processos, simulação e otimização multicritério para o agendamento cirúrgico. Essa abordagem automatiza parcialmente a geração de modelos e utiliza indicadores de eficiência para suportar decisões em tempo reduzido. Contudo, sua aplicação ainda é restrita ao domínio da saúde, carecendo de generalização para outros tipos de processos.

Na área industrial, trabalhos como (WUENNENBERG; WEGERICH; FOTTNER, 2023) propõem pipelines que unem simulação e mineração de processos em sistemas logísticos internos. Esses modelos exploram o uso de simulação para geração de dados sintéticos, que são posteriormente minerados para verificação de conformidade e detecção de gargalos. Em paralelo, pesquisas em mineração subterrânea (BRZYCHCZY; ŻUBER; AALST, 2024) e simulação modular (MENG et al., 2024) também avançam na integração entre dados de sensores, abstração de eventos e análise preditiva, embora com foco em contextos físicos específicos.

Em síntese, as soluções atuais demonstram o potencial da integração entre PM e DES, mas permanecem limitadas quanto à automação de ponta a ponta e à adaptabilidade entre diferentes domínios. O presente trabalho propõe evoluir essas abordagens por meio de um gerador de modelos de simulação automatizado e generalizável, reduzindo o esforço técnico necessário e ampliando o alcance da análise preditiva em processos de decisão operacional de curto prazo.

1.5 Objetivos

1.5.1 Objetivo Geral

Desenvolver um **gerador de modelos de simulação baseado em mineração de processos** para apoiar a **tomada de decisão no curto prazo**, capaz de criar automaticamente modelos de simulação a partir de logs de eventos, reduzindo o esforço humano e o tempo necessário para a modelagem de sistemas complexos.

1.5.2 Objetivos Específicos

Para alcançar o objetivo geral, este trabalho busca:

- **Analisar** métodos de integração PM-DES, documentando grau de automação, precisão de modelos e tempo de execução de pelo menos 3 abordagens existentes;
- **Projetar** arquitetura modular que suporte: (a) leitura automática de logs XES, (b) descoberta de processos via Inductive Miner, (c) extração estatística de distribuições de tempos (normal, lognormal, exponencial), (d) identificação automática de recursos, (e) geração de código SimPy parametrizado;
- **Validar** em estudo de caso hospitalar real: demonstrar correspondência entre comportamento simulado e dados históricos com fitness score $\geq 0,80$ e precisão $\geq 0,75$;
- **Avaliar** potencial de generalização através de testes em pelo menos 2 domínios adicionais (administrativo/logístico) e documentar limitações técnicas identificadas.

1.6 Contribuições Principais

Este trabalho apresenta as seguintes contribuições:

- **Arquitetura automatizada end-to-end:** Pipeline integrado que transforma logs XES em modelos DES parametrizados sem intervenção manual, eliminando lacuna de 2-3 semanas típica de modelagem convencional para execução em minutos.
- **Protocolo de parametrização estatística:** Método sistemático para extração automática de distribuições de tempos, identificação de recursos e padrões de decisão diretamente de event logs, garantindo reprodutibilidade e consistência.
- **Framework generalizável cross-domain:** Solução testada em contexto hospitalar mas aplicável a manufatura, logística, administração e serviços, desde que dados estejam estruturados em padrão XES.

- **Validação com dados reais:** Protótipo validado com logs históricos de processos cirúrgicos, demonstrando fitness e precisão compatíveis com literatura, comprovando viabilidade operacional.
- **Redução de esforço humano:** Democratização do acesso a simulação como ferramenta de apoio à decisão, ampliando capacidade analítica de organizações sem expertise em modelagem formal.

1.7 Justificativa

A literatura demonstra potencial da integração PM-DES (MARUŞTER; BEEST, 2009; WUENNENBERG; WEGERICH; FOTTNER, 2023; FERRONATO, 2022), porém com dependência de etapas manuais em modelagem, parametrização e calibração. Este trabalho avança teoricamente ao propor método completamente automatizado, com protocolo replicável e reprodutível, ampliando o corpo de conhecimento sobre como transformar dados históricos em modelos preditivos sem perda de precisão. Contribui para reconciliar a natureza descritiva de PM com a capacidade preditiva de DES em contexto operacional.

O protótipo representa avanço em relação a PM4SOS (FERRONATO, 2022) e metodologias similares. Enquanto trabalhos precedentes (MARUŞTER; BEEST, 2009; WUENNENBERG; WEGERICH; FOTTNER, 2023) requerem 2-3 semanas de modelagem manual especializada, esta solução automatiza o pipeline completo em minutos. Diferencia-se por: (i) eliminação de intervenção manual, (ii) generalizabilidade cross-domain (não restrita a saúde), (iii) parametrização estatística sistemática com distribuições ajustadas aos dados (normal, lognormal, exponencial), (iv) integração em ferramenta única e acessível.

Organizações modernas enfrentam pressão por decisões ágeis. Dados operacionais abundam, mas análises demoram semanas. A solução proposta reduz este ciclo para minutos, viabilizando uso operacional cotidiano. Validação com dados reais de processos cirúrgicos comprova viabilidade e aplicabilidade em contextos operacionais complexos (FERRONATO, 2022). Potencial de aplicação: saúde, manufatura, logística, administração — qualquer domínio com logs estruturados XES. Amplia a capacidade analítica organizacional sem exigir expertise em modelagem formal, democratizando acesso a ferramentas de simulação.

2 Referencial Teórico

Este capítulo apresenta os fundamentos conceituais e técnicos que sustentam o desenvolvimento de um gerador de modelos de simulação orientado por mineração de processos. São abordados desde os conceitos fundamentais de mineração de processos até a integração com simulação de eventos discretos e indicadores de desempenho operacional aplicados a ambientes hospitalares.

2.1 Mineração de Processos

A mineração de processos é uma disciplina que une conceitos de mineração de dados e de gerenciamento de processos de negócio com o objetivo de extrair conhecimento útil a partir de registros de eventos provenientes de sistemas de informação. Segundo van der Aalst (AALST, 2016), a mineração de processos tem como propósito descobrir, monitorar e aprimorar processos reais com base nos dados efetivamente registrados, constituindo uma ponte entre a análise orientada a dados e a modelagem formal de processos.

Com a crescente digitalização das operações empresariais e o avanço de sistemas como ERPs, CRMs e sistemas de controle de manufatura, passou a ser possível registrar detalhadamente cada etapa executada em um processo. Esses registros, conhecidos como *event logs*, formam a base para a aplicação de técnicas de mineração de processos. Cada evento registrado representa a execução de uma atividade pertencente a um caso e contém atributos como o nome da atividade, o identificador do caso, o responsável (*resource*) e o carimbo de tempo (*timestamp*). A estrutura desses logs permite a reconstituição da sequência de atividades e o estudo do comportamento do processo real (MARUŞTER; BEEST, 2009).

A mineração de processos é composta por três grandes categorias de técnicas (AALST, 2016):

1. **Descoberta de processos (*Process Discovery*)**: tem como objetivo gerar automaticamente um modelo de processo a partir de um log de eventos, sem conhecimento prévio do fluxo. O modelo resultante pode ser representado em diferentes notações, como Redes de Petri, BPMN (*Business Process Model and Notation*) ou Árvores de Processo (*Process Trees*).

2. **Verificação de conformidade (*Conformance Checking*)**: consiste em comparar um modelo de processo pré-existente com um log de eventos real, avaliando a aderência entre o comportamento observado e o comportamento esperado. Essa comparação fornece métricas como *fitness* e *precision*.

3. **Aprimoramento de modelos (*Enhancement*)**: busca enriquecer modelos existentes com informações adicionais provenientes dos logs, como tempos médios de execução, gargalos, desvios e uso de recursos, permitindo a análise de desempenho e a detecção de oportunidades de otimização.

O ciclo de mineração de processos, descrito no *Process Mining Manifesto* da IEEE Task Force on Process Mining (AALST, 2016), enfatiza que a aplicação prática da técnica depende da disponibilidade e da qualidade dos logs de eventos. Logs incompletos, inconsistentes ou mal formatados comprometem a acurácia dos modelos descobertos. Nesse sentido, Kherbouche et al. (KHERBOUCHE; LAGA; MASSE, 2020) destacam a importância da avaliação da qualidade dos logs, propondo métricas de *completude*, *consistência* e *complexidade* antes da aplicação das técnicas de mineração.

Em termos de arquitetura, um sistema de mineração de processos segue um fluxo básico: coleta de dados, pré-processamento, mineração propriamente dita e análise dos resultados. Durante a coleta, os logs podem ser extraídos de sistemas como SAP, Oracle, ou de bancos de dados customizados. O pré-processamento envolve a limpeza e padronização dos dados, incluindo a identificação correta de casos e atividades. Em seguida, algoritmos como *Alpha Miner*, *Heuristic Miner* e *Inductive Miner* são aplicados para a descoberta do modelo de processo (LEEMANS; FAHLAND; AALST, 2013).

Os resultados podem ser apresentados em notações formais, como Redes de Petri, ou em linguagens mais visuais, como BPMN. O uso de ferramentas especializadas, como o *ProM Framework* e a biblioteca Python PM4Py, facilita essa análise e integração com outros métodos, como a simulação de eventos discretos (FERRONATO; SCALABRIN, 2021).

O padrão *eXtensible Event Stream* (XES), definido pela IEEE (IEEE Computational Intelligence Society, 2010), estabelece a estrutura de logs de eventos para garantir interoperabilidade entre ferramentas e consistência na troca de dados. Cada log em XES é composto por uma coleção de *traces* (casos), e cada *trace* contém uma sequência ordenada de *events*. Essa padronização é essencial para o sucesso de frameworks modernos de mineração de processos e simulação integrada.

Portanto, a mineração de processos se consolida como uma abordagem essencial para compreender, auditar e melhorar processos organizacionais em ambientes baseados em dados. Quando combinada com simulação e análise estatística, ela se torna uma ferramenta poderosa para suporte à decisão, otimização operacional e melhoria contínua de processos complexos, como os hospitalares e logísticos.

2.2 Inductive Miner

Entre os diversos algoritmos de descoberta de processos disponíveis, o *Inductive Miner* (IM), proposto por Leemans, Fahland e van der Aalst (LEEMANS; FAHLAND; AALST, 2013), é um dos mais relevantes e amplamente utilizados na literatura e em ferramentas modernas de mineração, como o *ProM* e o `PM4Py`.

Diferentemente de abordagens anteriores, como o *Alpha Miner* e o *Heuristic Miner*, o Inductive Miner foi projetado para garantir propriedades formais no modelo resultante, como a *soundness* (correção comportamental) e a estrutura hierárquica em blocos. Essas propriedades asseguram que o modelo possa ser executado sem estados mortos ou impasses, além de facilitar sua conversão em linguagens formais como Redes de Petri e Árvores de Processo.

O princípio fundamental do algoritmo baseia-se na decomposição recursiva do log de eventos. O IM analisa as relações de precedência e causalidade entre atividades e divide o log em sublogs coerentes, representando fragmentos independentes do processo. Cada sublog é então minerado de forma isolada, e os resultados são combinados em uma estrutura hierárquica que reflete a composição lógica das atividades.

Uma das vantagens práticas do Inductive Miner é sua compatibilidade direta com representações de simulação. Ao produzir modelos formalmente corretos e livres de inconsistências estruturais, o IM facilita a transformação automática em modelos de simulação baseados em Redes de Petri, como destacado por Ferronato (FERRONATO; SCALABRIN, 2021). Essa característica é essencial para o desenvolvimento de sistemas como o *PM2Sim*, que automatiza a criação de modelos de simulação de eventos discretos.

2.3 Redes de Petri

As Redes de Petri constituem um formalismo matemático e gráfico amplamente utilizado para modelar, analisar e simular sistemas de eventos discretos. Propostas originalmente por Carl Adam Petri na década de 1960 e formalizadas por Peterson (PETERSON, 1981), essas redes oferecem uma base rigorosa para a representação de processos dinâmicos caracterizados por concorrência, sincronização, conflito e causalidade — propriedades típicas de sistemas produtivos, logísticos e hospitalares.

Uma Rede de Petri é definida formalmente como uma tupla $N = (P, T, F)$, onde:

- P representa o conjunto de lugares (*places*);
- T representa o conjunto de transições (*transitions*);
- $F \subseteq (P \times T) \cup (T \times P)$ é o conjunto de arcos que conectam lugares e transições.

Os lugares simbolizam condições ou estados do sistema, enquanto as transições representam eventos ou atividades que modificam esses estados. A dinâmica do sistema é descrita por meio da movimentação de fichas (*tokens*) entre os lugares. O conjunto de tokens em um dado instante define a marcação atual da rede (*marking*), representando o estado do processo. Quando todas as condições de disparo de uma transição são satisfeitas, ela se torna habilitada e pode ser executada, consumindo e produzindo tokens conforme o fluxo definido em F. Essa semântica de disparo possibilita a modelagem de sistemas paralelos e assíncronos de forma intuitiva e precisa.

Segundo Peterson (PETERSON, 1981), uma das principais vantagens das Redes de Petri é a possibilidade de realizar análises formais de propriedades do sistema modelado, como:

- **Alcançabilidade** (*Reachability*): determina quais estados podem ser atingidos a partir da marcação inicial.
- **Viveza** (*Liveness*): garante que nenhuma transição se torne permanentemente inativa, evitando estados mortos.
- **Conservação** (*Boundedness*): assegura que o número de tokens em cada lugar permanece finito, prevenindo explosões de estados.
- **Deadlock-freedom**: certifica que o sistema não entra em impasse completo.

Essas propriedades são fundamentais para a verificação de correção comportamental (*soundness*) em modelos de processos descobertos via mineração, assegurando que o modelo é executável e que todo caso pode ser concluído corretamente (AALST, 2016).

No contexto da mineração de processos, as Redes de Petri são amplamente utilizadas como formalismo intermediário para representar os modelos extraídos de logs de eventos. O *Inductive Miner*, por exemplo, gera diretamente uma Rede de Petri *sound* e *block-structured* (LEEMANS; FAHLAND; AALST, 2013), permitindo tanto a verificação de conformidade quanto a execução simulada do processo. Essa característica é essencial para a integração entre mineração e simulação de eventos discretos, pois possibilita a tradução direta de modelos minerados em estruturas simuláveis.

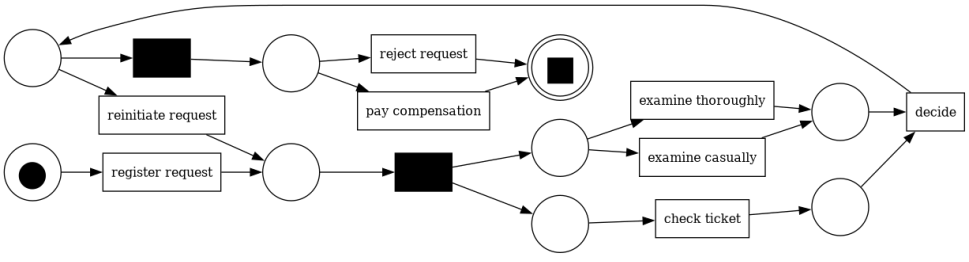

Figura 1 – Exemplo de Rede de Petri representando um processo de negócio. Fonte: Autor (2025)

Na Figura 1, os elementos da Rede de Petri são representados da seguinte forma: os círculos representam *lugares* (places), que indicam estados ou condições do processo; os retângulos representam *transições* (transitions), que correspondem a atividades ou eventos que podem ocorrer; os círculos com pontos pretos internos representam *tokens* (marcas), que indicam a presença de uma condição ou o estado atual do processo; e os círculos com contorno destacado e tokens internos representam lugares marcados, indicando estados ativos no momento atual da execução.

Ferronato (FERRONATO; SCALABRIN, 2021) destaca que as Redes de Petri desempenham um papel central na integração entre mineração e simulação. Em seu framework *PM2Sim*, as redes extraídas a partir de logs de eventos são automaticamente convertidas em modelos de simulação de eventos discretos implementados em Python. Essa conversão permite avaliar métricas como tempo de ciclo, gargalos e utilização de recursos, transformando os modelos minerados em instrumentos de suporte à decisão operacional.

Em síntese, as Redes de Petri constituem uma linguagem formal robusta e expressiva para representar o comportamento de sistemas reais. Sua adoção no contexto da mineração de processos garante não apenas a fidelidade comportamental dos modelos gerados, mas também sua viabilidade para análises de desempenho e simulação, consolidando-se como elo fundamental entre a teoria de processos e a prática da modelagem computacional.

2.4 Simulação de Eventos Discretos

A simulação de eventos discretos é uma técnica de modelagem computacional amplamente utilizada para representar sistemas dinâmicos em que o estado do sistema muda apenas em pontos discretos no tempo. Cada mudança é provocada por um evento, que ocorre em um instante específico e representa uma transição de estado. Essa abordagem é amplamente empregada em áreas como manufatura, logística, saúde e serviços, onde os processos são compostos por atividades sequenciais, paralelas e dependentes de recursos.

A DES permite reproduzir o comportamento de sistemas complexos sem a necessidade de interferir em seu ambiente real, possibilitando análises de desempenho, previsão de gargalos e avaliação de cenários alternativos. Diferentemente da simulação contínua, que modela fenômenos em tempo contínuo por meio de equações diferenciais, a simulação discreta descreve processos orientados por eventos, sendo, portanto, ideal para a representação de fluxos de trabalho e processos empresariais.

Em termos formais, um modelo de simulação de eventos discretos é composto por três elementos básicos:

- **Entidades**: representam os objetos que transitam pelo sistema (por exemplo, pacientes, ordens de serviço ou produtos).

- **Recursos**: correspondem aos elementos que executam as atividades (como profissionais, máquinas ou salas cirúrgicas).
- **Eventos**: são as ocorrências que alteram o estado do sistema, como o início ou o término de uma atividade.

Esses componentes são orquestrados por um *motor de simulação* (*simulation engine*) que mantém um relógio lógico e agenda os próximos eventos a serem processados. Cada evento executado pode gerar novos eventos futuros, modificando o estado do sistema e permitindo a evolução temporal do modelo (LIU, 2015).

A integração entre simulação e mineração de processos vem sendo explorada nos últimos anos. Liu (LIU, 2015) demonstrou que os modelos extraídos via *process mining* podem ser automaticamente convertidos em modelos de simulação de eventos discretos, reduzindo o esforço de modelagem e aumentando a precisão analítica. Essa integração é especialmente relevante em contextos onde a dinâmica do sistema se altera frequentemente, exigindo atualizações rápidas de modelos e previsões.

Nesse contexto, Ferronato e Scalabrin (FERRONATO; SCALABRIN, 2021) desenvolveram o framework *PM2Sim*, que automatiza a criação de modelos de simulação a partir de logs de eventos reais. O sistema identifica atividades, durações, tempos de espera e recursos envolvidos, transformando essas informações em um modelo DES implementado em Python com a biblioteca `SimPy`. Essa biblioteca fornece uma estrutura orientada a processos, baseada em geradores, que permite modelar entidades como processos que interagem em um ambiente temporal discreto. A `SimPy` oferece ainda suporte à criação de filas, controle de recursos, eventos simultâneos e coleta de estatísticas de desempenho, tornando-a ideal para aplicações em mineração de processos e análise operacional.

Além disso, o framework *PM2Sim* utiliza distribuições estatísticas ajustadas por meio de técnicas de *fitting* (adequação), com base em bibliotecas como `NumPy` e `SciPy`, permitindo representar o comportamento dos tempos de execução das atividades. O resultado é um modelo de simulação que reflete o comportamento observado nos logs, possibilitando a análise de métricas como tempo de ciclo, utilização de recursos e identificação de gargalos.

De acordo com Wuennenberg et al. (WUENNENBERG; WEGERICH; FOTTNER, 2023), a combinação de mineração de processos e simulação discreta é particularmente útil em ambientes industriais e logísticos, onde a qualidade dos dados e a variabilidade operacional representam desafios significativos. Nesse sentido, a DES serve como ferramenta de validação e experimentação para modelos minerados, permitindo testar hipóteses e prever o impacto de mudanças estruturais antes de sua implementação no ambiente real.

2.5 Análise Estatística

A etapa de análise estatística é fundamental para a construção de modelos de simulação realistas e coerentes com os processos observados. Após a descoberta do modelo de processo e a extração das informações de desempenho a partir dos logs de eventos, é necessário realizar o ajuste estatístico dos parâmetros temporais, como durações de atividades, tempos de espera e intervalos entre eventos. Esses parâmetros são essenciais para que o modelo de simulação reflita adequadamente a variabilidade e o comportamento estocástico do sistema.

Segundo van der Aalst (AALST, 2016), a mineração de processos deve ser vista como uma disciplina orientada por dados (*data-driven*), e o sucesso de suas aplicações depende da correta interpretação das distribuições de tempo, frequência e desempenho que emergem dos registros de eventos. O uso de técnicas estatísticas complementa a fase de descoberta, permitindo transformar modelos descritivos em modelos quantitativos capazes de realizar simulações e análises preditivas.

Ferronato e Scalabrin (FERRONATO; SCALABRIN, 2021) destacam que o uso de distribuições estatísticas é um dos pilares do framework *PM2Sim*, que automatiza a criação de modelos de simulação a partir de logs reais. No sistema proposto, os tempos de execução das atividades e os intervalos entre eventos são ajustados a partir de funções de probabilidade clássicas, como Normal, Log-Normal e Exponencial. Essa parametrização é obtida por meio do ajuste de distribuições (*distribution fitting*), realizado com base nos dados históricos de cada atividade registrada no log. O processo envolve a estimativa dos parâmetros das distribuições e a verificação de aderência dos dados observados, assegurando a representatividade do modelo.

Para realizar esses ajustes, bibliotecas estatísticas como `NumPy` e `SciPy` são utilizadas no ambiente Python, permitindo estimar as distribuições de probabilidade e calcular medidas como média, variância, desvio padrão e coeficiente de variação. Além disso, testes de aderência, como o de Kolmogorov–Smirnov, são empregados para validar se os dados amostrais seguem adequadamente a distribuição teórica selecionada. Esse procedimento garante que os tempos de simulação não apenas reproduzam as médias históricas, mas também capturem a variabilidade natural do processo (LIU, 2015).

O resultado da análise estatística é incorporado diretamente aos parâmetros do modelo de simulação, alimentando o mecanismo de eventos discretos com tempos amostrados de distribuições probabilísticas. Essa abordagem permite representar o comportamento dinâmico e imprevisível dos processos reais, algo essencial em ambientes sujeitos a variações operacionais, como hospitais e sistemas logísticos. Conforme observado por Leemans et al. (LEEMANS; FAHLAND; AALST, 2013), a precisão dos tempos e a correta modelagem da frequência das atividades impactam diretamente a capacidade do modelo minerado em reproduzir o fluxo real dos eventos.

Assim, a análise estatística atua como uma ponte entre a descoberta de processos e

a simulação de eventos discretos, traduzindo dados empíricos em parâmetros quantitativos para o modelo. Essa integração garante que a simulação gerada preserve as características estatísticas do processo original, permitindo a avaliação confiável de métricas de desempenho e a experimentação de cenários alternativos de operação.

2.6 Métricas de Qualidade de Modelos

A qualidade dos modelos descobertos por meio da mineração de processos é um fator determinante para a confiabilidade das análises subsequentes e, principalmente, para o uso desses modelos como base para simulações. A avaliação sistemática da qualidade garante que o modelo minerado não apenas represente corretamente o comportamento histórico, mas também seja capaz de generalizar o comportamento futuro e sustentar decisões operacionais baseadas em evidências.

De acordo com van der Aalst (AALST, 2016), um modelo de processo deve ser avaliado em múltiplas dimensões de qualidade, de modo a equilibrar precisão, generalização e simplicidade. As principais métricas utilizadas na literatura são: *fitness*, *precision*, *generalization* e *simplicity*. Essas métricas, amplamente adotadas em ferramentas como o *ProM Framework* e a `PM4Py`, compõem a base dos algoritmos de verificação de conformidade (*conformance checking*), responsáveis por comparar o comportamento observado nos logs de eventos com o comportamento previsto pelo modelo minerado.

- **Fitness**: avalia o quanto o modelo reproduz o comportamento observado no log. Um modelo com alto *fitness* consegue reproduzir todas as sequências válidas de atividades sem apresentar falhas de execução.
- **Precision**: mede o nível de restrição do modelo, penalizando comportamentos não observados nos dados. Modelos excessivamente permissivos tendem a apresentar alta flexibilidade, mas baixa precisão.
- **Generalization**: representa a capacidade do modelo de capturar variações plausíveis do processo, evitando o sobreajuste aos dados históricos.
- **Simplicity**: reflete o grau de complexidade estrutural do modelo; modelos excessivamente complexos, embora precisos, dificultam a interpretação e a manutenção (AALST et al., 2012).

O *Process Mining Manifesto* (AALST et al., 2012) destaca a importância de equilibrar essas dimensões, pois a busca por um modelo com *fitness* perfeito pode levar à perda de generalização, e vice-versa. A maturidade da área de mineração de processos se reflete justamente na capacidade de lidar com esse compromisso entre precisão e abstração. Esse equilíbrio é

particularmente relevante quando os modelos descobertos serão empregados em simulações, uma vez que comportamentos não observados ou superajustados podem comprometer a validade dos resultados simulados.

Liu (LIU, 2015) ressalta que a integração entre mineração e simulação exige não apenas um modelo logicamente consistente, mas também estatisticamente representativo. A validação entre logs reais e logs sintéticos simulados permite medir a aderência entre ambos, utilizando métricas de distância e similaridade, como a *edit distance*. Essa abordagem garante que a simulação não apenas reproduza o fluxo de atividades, mas também mantenha coerência temporal e probabilística com o processo original.

Kherbouche et al. (KHERBOUCHE; LAGA; MASSE, 2020) complementam essa perspectiva ao argumentar que a qualidade do modelo está diretamente relacionada à qualidade do log de eventos. Logs incompletos, redundantes ou ruidosos geram modelos com baixa precisão e menor confiabilidade. Por isso, o controle da qualidade dos logs — avaliado por dimensões como *completude*, *consistência*, *acurácia* e *complexidade* — é pré-requisito para a obtenção de métricas significativas de *fitness* e *precision*.

Portanto, as métricas de qualidade de modelos constituem um elo crítico entre mineração e simulação. Elas fornecem os indicadores necessários para validar a representatividade, robustez e aplicabilidade do modelo descoberto. Um modelo de processo só é efetivamente útil quando combina boa aderência aos dados históricos com capacidade de generalização para novos cenários — característica essencial para o uso em ambientes de tomada de decisão e otimização operacional.

2.7 Geração de Logs Sintéticos

A geração de logs sintéticos é uma etapa estratégica na integração entre mineração de processos e simulação de eventos discretos. Essa técnica permite criar registros artificiais de eventos que preservam as propriedades estruturais e estatísticas de logs reais, viabilizando experimentos controlados, testes de desempenho e validação de modelos sem a necessidade de utilizar dados sensíveis ou restritos. Em termos práticos, os logs sintéticos servem como uma representação simulada do comportamento observado, permitindo avaliar a fidelidade dos modelos minerados e a confiabilidade das previsões operacionais.

Segundo van der Aalst (AALST, 2016), a utilização de logs artificiais é essencial para verificar se o modelo de processo descoberto é capaz de reproduzir o comportamento do sistema real. Essa comparação é comumente feita por meio de técnicas de *conformance checking*, nas quais o log sintético gerado pelo modelo é confrontado com o log original, medindo-se métricas como *fitness* e *precision*. Essa abordagem é particularmente útil em ambientes dinâmicos, onde a estrutura do processo pode mudar com frequência, como na área hospitalar e em sistemas

logísticos.

Ferronato (FERRONATO; SCALABRIN, 2021) propõe no framework *PM2Sim* um processo automatizado de geração de logs sintéticos a partir de modelos descobertos via mineração de processos. O sistema transforma o modelo minerado — geralmente representado como uma Rede de Petri — em um modelo de simulação implementado em Python por meio da biblioteca `SimPy`. Durante a execução da simulação, eventos são registrados em formato XES (*eXtensible Event Stream*), mantendo a compatibilidade com ferramentas de mineração como PM4Py e ProM. Essa estratégia permite a retroalimentação do ciclo de mineração, criando uma integração contínua entre descoberta, simulação e validação.

Augusto et al. (AUGUSTO et al., 2016) reforçam essa importância ao aplicar a integração entre mineração de processos e simulação em um contexto clínico. Os autores desenvolveram uma metodologia para simular fluxos de pacientes com base em logs hospitalares nacionais, combinando mineração e simulação. Essa abordagem permitiu a criação de logs sintéticos de trajetórias clínicas, utilizados para avaliar o impacto de decisões médicas e políticas de gestão sobre taxas de mortalidade e custos. Os resultados evidenciam o potencial da geração de logs sintéticos como ferramenta de experimentação em ambientes de alta complexidade e variabilidade.

A geração de dados artificiais também desempenha um papel importante na análise de eficiência operacional de ambientes hospitalares. O estudo de Protil et al. (STROPARO; BICHINHO; PROTIL, 2004) sobre a ocupação de centros cirúrgicos demonstrou que o uso de modelos simulados possibilita identificar desperdícios de tempo e gargalos no agendamento de cirurgias. Embora o trabalho não empregue mineração de processos, ele antecipa a importância da simulação como meio de geração de dados para planejamento e otimização de recursos — uma função equivalente à dos logs sintéticos em contextos modernos.

Outro aspecto relevante é a qualidade dos logs gerados. Conforme Kherbouche et al. (KHERBOUCHE; LAGA; MASSE, 2020), a utilidade dos logs sintéticos depende da fidelidade com que reproduzem as características estatísticas, temporais e comportamentais dos registros reais. Logs incompletos, redundantes ou inconsistentes podem comprometer a avaliação do modelo.

A mineração fornece o modelo descritivo; a simulação, o ambiente de experimentação; e os logs sintéticos, o instrumento de validação. Esse ciclo permite aprimorar continuamente o modelo de processo, ajustando suas propriedades comportamentais e estatísticas com base em dados observados e simulados. Resumidamente, a geração de logs sintéticos transforma o modelo minerado em uma ferramenta viva de aprendizado e decisão, capaz de antecipar cenários e apoiar a otimização operacional em tempo reduzido.

2.8 Indicadores de Desempenho em Simulação

A avaliação de desempenho é um componente essencial em sistemas que utilizam simulação de eventos discretos para otimização de processos. Em simulação, indicadores tradicionais incluem tempo de ciclo (*lead time*), utilização de recursos, capacidade (*throughput*) e eficiência operacional. Essas métricas permitem comparar cenários alternativos e validar se mudanças estruturais geram os ganhos esperados.

No contexto hospitalar específico, Souza, Vaccaro e Lima (SOUZA; VACCARO; LIMA, 2020) propuseram o indicador *Operating Room Effectiveness* (ORE), adaptado do conceito de *Overall Equipment Effectiveness* (OEE) da manufatura enxuta. O ORE é calculado pelo produto de três dimensões — Disponibilidade, Desempenho e Qualidade — permitindo decompor perdas operacionais em categorias mensuráveis: falhas de equipamento, setup, variações de tempo, cancelamentos e reintervenções.

O ORE foi concebido para fornecer uma visão integrada da efetividade operacional de salas cirúrgicas, permitindo identificar e classificar perdas operacionais de forma sistemática. Diferentemente de indicadores isolados como taxa de ocupação, tempo de cancelamentos ou resultados financeiros, o ORE oferece uma métrica global que correlaciona múltiplas dimensões do desempenho operacional.

A base conceitual do ORE provém do OEE, desenvolvido por Nakajima (NAKAJIMA, 1988) no contexto da Manutenção Produtiva Total (TPM). O OEE é calculado pelo produto de três índices: Disponibilidade, Desempenho e Qualidade. Essa estrutura multiplicativa permite identificar onde ocorrem as principais perdas do sistema produtivo. A adaptação para o contexto hospitalar mantém essa estrutura, mas reinterpreta as perdas considerando as especificidades do fluxo cirúrgico.

A fórmula geral do ORE é expressa por:

$$\text{ORE} = \text{Disponibilidade} \times \text{Desempenho} \times \text{Qualidade} \tag{2.1}$$

Alternativamente, o ORE pode ser calculado como:

$$\text{ORE} = \frac{\text{Tempo Total de Valor Agregado (TTAV)}}{\text{Tempo Total Disponível (TTA)}} \tag{2.2}$$

Cada componente do indicador é detalhado nas subseções seguintes.

2.8.1 Dimensões e Perdas Operacionais

O ORE classifica as perdas operacionais em três classes principais, totalizando sete tipos distintos de perdas. Essa classificação é fundamental para direcionar ações de melhoria de forma precisa e mensurável.

2.8.1.1 Perdas de Disponibilidade

A Disponibilidade mede a fração do tempo programado que efetivamente foi utilizado para atividades cirúrgicas, descontando perdas relacionadas ao planejamento e preparação das salas. É calculada como:

$$\text{Disponibilidade} = \frac{\text{Tempo Total Programado (TTS)}}{\text{Tempo Total Disponível (TTA)}} \tag{2.3}$$

As perdas de disponibilidade incluem:

- **Falha de Equipamento**: tempo de inatividade causado por quebras ou manutenções não programadas de equipamentos cirúrgicos, anestésicos ou de suporte.
- **Setup e Paradas Programadas**: tempo necessário para limpeza, preparação e esterilização da sala entre procedimentos. Este tempo é estrutural e inevitável, mas pode ser otimizado através de padronização e melhoria de processos.
- **Não Agendamento**: tempo ocioso em turnos reservados a cirurgiões específicos, mas sem cirurgias programadas. Esta perda é particularmente relevante em hospitais universitários onde há alocação pré-definida de blocos cirúrgicos por equipe médica.

2.8.1.2 Perdas de Desempenho

O Desempenho mede a fração do tempo programado que foi efetivamente utilizado, descontando variações em relação ao planejado e cancelamentos. É calculado como:

$$\text{Desempenho} = \frac{\text{Tempo Total Utilizado (TTU)}}{\text{Tempo Total Programado (TTS)}} \tag{2.4}$$

As perdas de desempenho incluem:

- **Pequenas Paradas**: interrupções momentâneas durante o procedimento cirúrgico causadas por problemas com equipamentos, suprimentos, falta de materiais ou quedas de energia.
- **Variação no Tempo Cirúrgico**: diferenças entre o tempo planejado e o tempo real de execução do procedimento. Cirurgias que excedem significativamente o tempo previsto podem causar cancelamentos em cascata das cirurgias subsequentes.
- **Cancelamentos**: cirurgias canceladas após o agendamento resultam em capacidade ociosa não recuperável. Cerca de 50% dos cancelamentos ocorrem por falta de condição clínica do paciente ou absenteísmo, indicando falhas na gestão do fluxo pré-operatório.

2.8.1.3 Perdas de Qualidade

A Qualidade mede a fração do tempo utilizado que efetivamente agregou valor ao paciente, descontando retrabalhos e reintervenções. É calculada como:

$$\text{Qualidade} = \frac{\text{Tempo Total de Valor Agregado (TTAV)}}{\text{Tempo Total Utilizado (TTU)}} \tag{2.5}$$

As perdas de qualidade incluem:

- **Reintervenções Cirúrgicas**: procedimentos que necessitaram ser refeitos devido a falhas técnicas, complicações evitáveis ou erros no procedimento inicial. Este tipo de perda é difícil de mensurar devido à complexidade de classificar reintervenções como evitáveis ou inerentes à evolução clínica do paciente.

2.8.2 ORE como Driver de Simulação

O uso do ORE transcende a função de medição passiva de desempenho. No contexto de simulação de processos, o ORE atua como um **indicador direcionador** que permite:

1. **Identificação de Oportunidades**: a decomposição do ORE nas sete categorias de perdas permite priorizar quais aspectos do processo apresentam maior potencial de ganho.
2. **Definição de Metas**: gestores podem estabelecer metas específicas de melhoria para cada dimensão do ORE (por exemplo, "reduzir cancelamentos em 30%"ou "reduzir tempo de setup em 5 minutos").
3. **Simulação de Cenários**: modelos de simulação podem ser parametrizados para alcançar valores-alvo de ORE, permitindo avaliar quais configurações operacionais (alocação de recursos, mudanças no cronograma, redução de variabilidade) são necessárias para atingir as metas estabelecidas.
4. **Ciclo de Melhoria Contínua**: a medição periódica do ORE permite avaliar o impacto de intervenções e ajustar estratégias de otimização de forma iterativa.

Souza et al. (SOUZA; VACCARO; LIMA, 2020) demonstraram que cenários hipotéticos de melhoria, quando simulados, podem estimar com precisão o impacto de intervenções antes de sua implementação. Por exemplo, a redução de 50% nos cancelamentos combinada com eliminação de tempos ociosos por não agendamento resultou em um ganho projetado de 31,6% no ORE, equivalente a aproximadamente 400 horas adicionais mensais de capacidade cirúrgica.

Os resultados apresentados por Souza et al. (SOUZA; VACCARO; LIMA, 2020) demonstraram ganhos de eficiência de 12% e economias anuais estimadas em US$400.000 após

a implementação do indicador e de ações de melhoria em um hospital universitário brasileiro. Tais resultados evidenciam o potencial do ORE como métrica de apoio à gestão operacional e à tomada de decisão em ambientes de alta complexidade.

Em trabalhos anteriores, Protil et al. (STROPARO; BICHINHO; PROTIL, 2004) já haviam explorado o uso de modelagem e simulação de sistemas para analisar a taxa de ocupação de centros cirúrgicos, apontando a importância da simulação como ferramenta para otimização de recursos hospitalares. A integração entre indicadores como o ORE e abordagens baseadas em mineração e simulação, conforme sugerido por Ferronato (FERRONATO; SCALABRIN, 2021), amplia a capacidade analítica desses sistemas, permitindo correlacionar métricas de eficiência com o comportamento real dos processos.

Dessa forma, o uso combinado de indicadores operacionais como o ORE e modelos minerados fornece uma visão quantitativa e dinâmica da eficiência hospitalar, permitindo avaliar e prever o impacto de decisões sobre produtividade, custos e qualidade dos serviços de saúde.

2.9 Padrão XES (eXtensible Event Stream)

O padrão *eXtensible Event Stream* (XES) foi desenvolvido pela *IEEE Task Force on Process Mining* com o objetivo de padronizar a representação de logs de eventos utilizados em mineração de processos. Formalizado pela norma IEEE 1849-2016 (IEEE Computational Intelligence Society, 2010), o XES define uma estrutura extensível e interoperável que permite armazenar e trocar informações sobre execuções de processos entre diferentes ferramentas e plataformas.

Cada log XES é composto por um conjunto de *traces*, que representam casos individuais de execução, e cada *trace* contém uma sequência ordenada de *events*, correspondentes às atividades executadas. Cada evento possui atributos obrigatórios — como nome da atividade, identificador do caso e carimbo de tempo — e opcionais, como recursos, custos ou anotações adicionais. Essa estrutura hierárquica assegura a consistência semântica dos dados e permite análises multi-perspectiva (fluxo, tempo e organização).

O Código 2.1 apresenta um trecho simplificado de um log XES que segue o padrão IEEE, ilustrando a estrutura de um *trace* (caso) com três eventos correspondentes a um processo de atendimento hospitalar.

Código 2.1 – Exemplo simplificado de log XES

```
<?xml version="1.0" encoding="UTF-8"?>
<log xes.version="1.0" xes.features="nested-attributes"
    xmlns="http://www.xes-standard.org/">
  <trace>
    <string key="concept:name" value="Case001"/>
```

```
    <event>
      <string key="concept:name" value="Admissao␣do␣Paciente"/>
      <date key="time:timestamp"
         value="2024-03-15T08:30:00.000+00:00"/>
      <string key="org:resource" value="Recepcao"/>
    </event>
    <event>
      <string key="concept:name" value="Avaliacao␣Medica"/>
      <date key="time:timestamp"
         value="2024-03-15T09:00:00.000+00:00"/>
      <string key="org:resource" value="Dr.␣Silva"/>
    </event>
    <event>
      <string key="concept:name" value="Alta␣Hospitalar"/>
      <date key="time:timestamp"
         value="2024-03-15T10:15:00.000+00:00"/>
      <string key="org:resource" value="Administracao"/>
    </event>
  </trace>
</log>
```

Nesse exemplo, o caso `Case001` representa a trajetória de um paciente desde a admissão até a alta hospitalar. Cada evento contém informações sobre a atividade executada (*concept:name*), o horário em que ocorreu (*time:timestamp*) e o recurso responsável (*org:resource*). A simplicidade e a extensibilidade do formato permitem que diferentes sistemas coletem e exportem dados compatíveis para posterior mineração.

3 Metodologia

Este capítulo apresenta a metodologia utilizada para o desenvolvimento do gerador de modelos de simulação baseado em mineração de processos. A abordagem metodológica foi estruturada em quatro etapas principais: análise automática de logs, mineração de processos, simulação de eventos discretos e validação de qualidade.

3.1 Visão Geral da Abordagem Metodológica

A metodologia proposta consiste em um pipeline sequencial de quatro etapas interdependentes:

1. **Análise Automática de Logs**: detecção de atributos e validação de compatibilidade
2. **Mineração de Processos**: extração do modelo formal e parâmetros estatísticos
3. **Simulação de Logs Sintéticos**: geração de casos baseada em eventos discretos
4. **Validação de Qualidade**: avaliação de similaridade e conformidade

3.2 Fluxo de Dados Detalhado

A Figura 2 ilustra o fluxo completo de dados através do sistema:

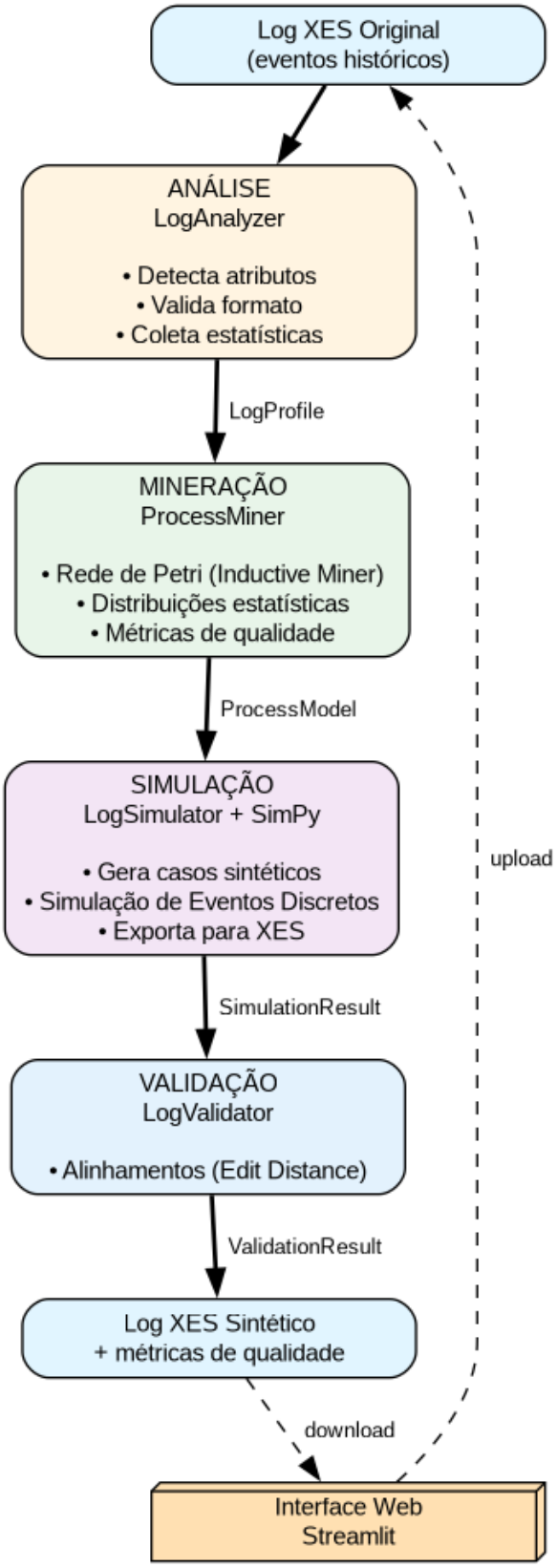

Figura 2 – Fluxo completo da metodologia. Fonte: Autor (2025)

Cada etapa produz artefatos específicos que alimentam a etapa seguinte, garantindo rastreabilidade e reprodutibilidade do processo. O sistema foi projetado para ser 100% genérico,

funcionando com logs de qualquer domínio organizacional que disponha de registros estruturados em formato XES.

3.3 Etapa 1: Análise Automática de Logs

3.3.1 Diagrama de Componente

A Figura 3 apresenta o diagrama de componente da Etapa 1. O componente central, `LogAnalyzer`, é responsável por realizar a análise automática e a extração de perfil de um log de eventos.

O componente recebe como entrada um **log de eventos** no formato XES, que encapsula os traces, eventos, atributos, timestamps e recursos do processo. A análise é parametrizada por uma **configuração** que define o modo de operação, como a ativação da auto-detecção de atributos e a validação do formato.

Internamente, o `LogAnalyzer` utiliza **métodos de análise** baseados em bibliotecas como PM4Py para realizar a detecção de padrões, análise de variantes e análise estatística. As principais tarefas executadas são a detecção automática da estrutura do log, a identificação das variantes do processo, a validação de compatibilidade com as etapas subsequentes e a extração de estatísticas descritivas.

Como resultado, o componente produz um **Perfil do Log** (`LogProfile`), um artefato estruturado que contém as características detectadas. Isso inclui métricas quantitativas (número de casos, eventos, atividades e recursos únicos), estruturais (variantes de processo, nomes dos atributos-chave identificados) e temporais (estatísticas de duração e frequência), além de uma verificação de compatibilidade para o pipeline.

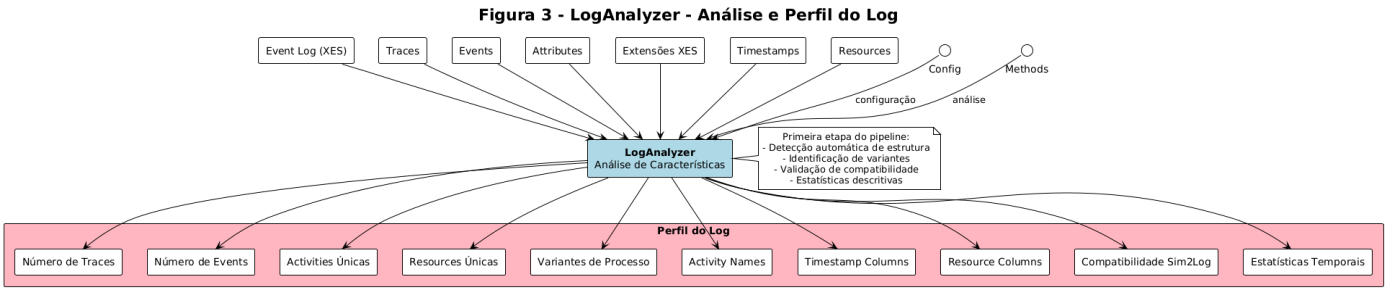

Figura 3 – Fase 1: Análise Automática de Logs. Fonte: Autor (2025)

3.3.2 Objetivo

Detectar automaticamente as características estruturais, temporais e operacionais do log de entrada, incluindo o cálculo de indicadores de eficiência operacional quando aplicável, asse-

gurando compatibilidade com diferentes domínios e formatos sem necessidade de configuração manual.

3.3.3 Justificativa

Logs de eventos variam significativamente entre domínios quanto à nomenclatura de atributos, presença de recursos organizacionais e convenções de registro temporal. A análise automática elimina a necessidade de parametrização manual e permite processamento agnóstico ao domínio, ampliando a aplicabilidade do sistema. Além da análise estrutural, a integração de indicadores de eficiência como o ORE permite quantificar perdas operacionais e direcionar a simulação para cenários de melhoria específicos.

3.3.4 Detecção de Atributos-Chave

O sistema implementa um mecanismo de detecção baseado em prioridades, testando sequencialmente diferentes convenções de nomenclatura até identificar os atributos obrigatórios:

- **Activity Key** (nome da atividade): `concept:name`, `Activity`, `activity`, `event`, `task`
- **Timestamp Key** (momento do evento): `time:timestamp`, `timestamp`, `Time`, `start_time`
- **Case ID Key** (identificador do caso): `concept:name`, `case_id`, `CaseID`, `Case`
- **Resource Key** (executor - opcional): `org:resource`, `resource`, `user`, `actor`

A ordem de prioridade baseia-se no padrão IEEE XES (IEEE Computational Intelligence Society, 2010) e em análise de datasets públicos do BPI Challenge e repositório 4TU.

Quando nenhum candidato da lista de prioridades é encontrado, o sistema aplica busca por palavras-chave nos nomes dos atributos. Se ainda assim a detecção falhar para atributos obrigatórios, uma exceção é lançada com lista dos atributos disponíveis para diagnóstico.

3.3.5 Coleta de Estatísticas Estruturais

Para cada log analisado, o sistema extrai as seguintes informações:

- Número total de casos (traces) e eventos
- Número de atividades únicas
- Distribuição de frequência das atividades

- Comprimento mínimo, máximo e médio dos traces
- Número de variantes do processo (sequências únicas de atividades)
- Distribuição de recursos por atividade (quando disponível)

Essas estatísticas são armazenadas em um objeto `LogProfile`, que serve como entrada para as etapas posteriores e permite análise exploratória dos dados antes da mineração.

3.3.6 Saída da Etapa

Um objeto `LogProfile` contendo:

- **Metadados estruturais**: número de traces, eventos, atividades únicas, variantes
- **Mapeamento de atributos**: activity_key, timestamp_key, case_id_key, resource_key (quando disponível)
- **Estatísticas descritivas**: frequências de atividades, comprimentos de traces, distribuição de variantes
- **Flags de compatibilidade**: has_resources, has_lifecycle, has_enriched_attributes
- **Informações de recursos**: mapeamento de recursos por atividade (quando disponível)

Nota sobre Métricas de Eficiência Operacional: O sistema possui um componente independente (`ORECalculator`) para cálculo de métricas de eficiência operacional específicas para processos hospitalares. Este componente é descrito em detalhes na Seção "Componente Independente: Cálculo de Métricas ORE"e pode ser invocado paralelamente ao pipeline principal quando o log contém atributos específicos do domínio hospitalar.

3.4 Etapa 2: Mineração de Processos

3.4.1 Diagrama de Componente

A Figura 4 apresenta o diagrama de componente da Etapa 2, detalhando o processo de mineração de processos e extração de parâmetros estatísticos.

O componente central `ProcessMiner` recebe como entradas o arquivo XES original, o objeto `LogProfile` produzido na Etapa 1 contendo os metadados do log, o parâmetro `variant_filter` configurado em 0.8 para filtragem de variantes, e uma flag `save_model_image` para geração opcional de visualização do modelo. O processamento interno é organizado em cinco fases sequenciais: (1) filtragem de variantes baseada em frequência, mantendo apenas as variantes que representam 80% dos casos totais para remoção

de ruído; (2) descoberta do modelo de processo utilizando o algoritmo Inductive Miner, que garante propriedades de soundness e sempre produz um modelo bem-formado; (3) conversão do Process Tree resultante para uma Rede de Petri formal com marcação inicial e final; (4) ajuste de distribuições estatísticas (Normal, Log-Normal e Exponencial) às durações observadas de cada atividade, utilizando o teste de Kolmogorov-Smirnov para seleção da distribuição com melhor aderência; (5) avaliação de qualidade do modelo através das métricas de fitness, precision e simplicity.

A saída produzida consiste em um objeto estruturado `ProcessModel` contendo: a Rede de Petri formal representada pela tupla (`net`, `im`, `fm`), onde `net` é a rede, `im` é a marcação inicial e `fm` é a marcação final; um dicionário de `ActivityStatistics` para cada atividade, incluindo duração média, desvio padrão, distribuição ajustada e seus parâmetros; métricas temporais globais como `arrival_rate` (taxa de chegada de casos), `dispersion_rate` e `median_duration`; métricas de qualidade do modelo (`fitness`, `precision`, `simplicity`); mapeamento de recursos por atividade quando disponível; e o `LogProfile` original para rastreabilidade. As bibliotecas PM4Py são utilizadas para implementação do Inductive Miner e cálculo de métricas de conformidade, enquanto SciPy (módulo `scipy.stats`) é empregado para ajuste e avaliação das distribuições estatísticas.

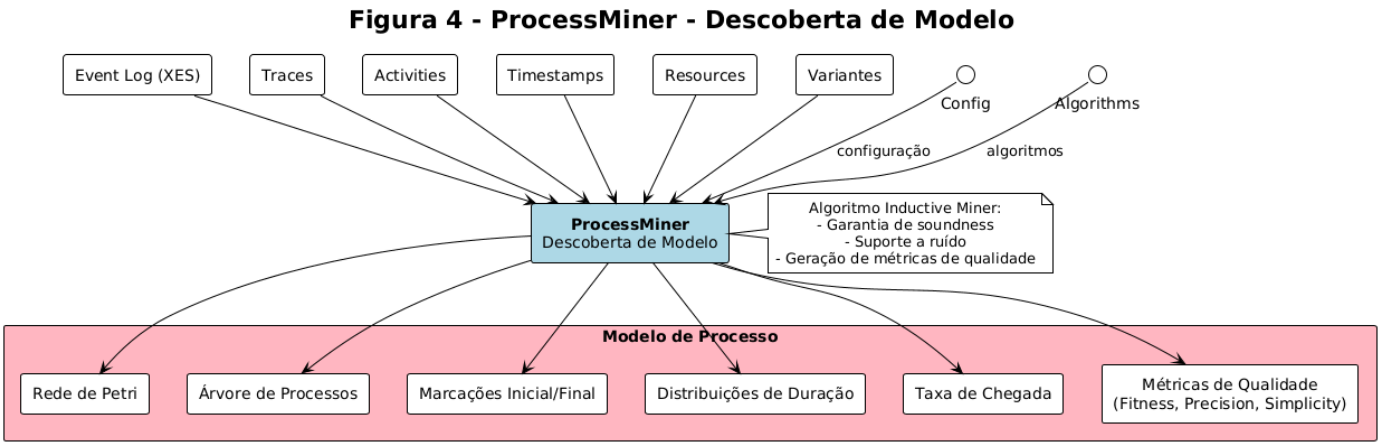

Figura 4 – Fase 2: Mineração de Processos. Fonte: Autor (2025)

3.4.2 Objetivo

Extrair o modelo formal do processo na forma de uma Rede de Petri e parametrizar suas características temporais e organizacionais para uso na simulação.

3.4.3 Filtragem de Variantes

3.4.3.1 Problema

Logs reais frequentemente contêm ruído, casos excepcionais e variantes raras que dificultam a descoberta de modelos representativos e podem levar a overfitting.

3.4.3.2 Solução

Aplicação de filtragem baseada em frequência, mantendo apenas as variantes que representam um percentual especificado dos casos totais.

3.4.3.3 Parâmetro Padrão

O sistema utiliza como padrão a retenção de 80% das variantes mais frequentes (`variant_filter=0.8`).

3.4.3.4 Justificativa

O valor de 80% fundamenta-se no Princípio de Pareto, onde aproximadamente 20% das variantes explicam 80% do comportamento observado (AALST, 2016). Este limiar oferece balance adequado entre cobertura comportamental e remoção de ruído.

3.4.4 Descoberta do Modelo de Processo

3.4.4.1 Algoritmo Selecionado

O sistema utiliza o **Inductive Miner** proposto por Leemans, Fahland e van der Aalst (LEEMANS; FAHLAND; AALST, 2013).

3.4.4.2 Justificativa da Escolha

A Tabela 1 apresenta comparação entre algoritmos de descoberta disponíveis:

Tabela 1 – Comparação entre algoritmos de descoberta de processos

Algoritmo	**Vantagens**	**Desvantagens**	**Decisão**
Alpha Miner	Simples, pioneiro	Não lida com ruído, loops	Não escolhido
Heuristic Miner	Tolerante a ruído	Modelos não-formais	Não escolhido
Inductive Miner	Soundness garantido, robusto a ruído, sempre produz modelo	Pode generalizar demais	**ESCOLHIDO**
Split Miner	Alta precisão	Requer tuning complexo	Não escolhido

O Inductive Miner foi escolhido por garantir três propriedades fundamentais:

1. **Soundness**: modelo sempre bem-formado, sem deadlocks
2. **Fitness**: modelo sempre capaz de reproduzir o log
3. **Completude**: sempre produz um modelo, mesmo com logs problemáticos

Além disso, o algoritmo gera modelos estruturados em blocos (block-structured), facilitando a conversão para formatos simuláveis.

3.4.4.3 Saída

Uma Rede de Petri formalmente definida como a tupla $N = (P, T, F)$, onde:

- P: conjunto de lugares (places)
- T: conjunto de transições (transitions)
- $F \subseteq (P \times T) \cup (T \times P)$: conjunto de arcos

Acompanhada de marcação inicial (M_0) e marcação final (M_f).

3.4.5 Extração de Estatísticas Temporais

3.4.5.1 Cálculo de Durações

Para cada atividade, o sistema calcula durações baseando-se em:

Casos com timestamp de conclusão:

$$duration = time : complete - time : timestamp \tag{3.1}$$

Casos sem timestamp de conclusão (maioria dos logs):

$$duration = timestamp(event_{i+1}) - timestamp(event_i) \tag{3.2}$$

3.4.6 Ajuste de Distribuições Estatísticas

3.4.6.1 Objetivo

Modelar a variabilidade realista das durações para simulação estocástica fiel ao comportamento observado.

3.4.6.2 Distribuições Candidatas

O sistema testa três distribuições de probabilidade:

1. **Normal (Gaussiana)**: para atividades com duração relativamente constante e variação simétrica
2. **Log-Normal**: para atividades com cauda longa à direita, comum em processos humanos
3. **Exponencial**: para tempos de espera e processos de chegada

3.4.7 Avaliação de Qualidade do Modelo

O sistema calcula três métricas complementares de qualidade (AALST, 2016):

3.4.7.1 Fitness (0-1, maior é melhor)

Definição: proporção de comportamento do log que o modelo consegue reproduzir.

Método: replay fitness baseado em alinhamentos.

Cálculo: O fitness utiliza token-based replay ou alignment-based conformance. A fórmula mais comum é:

$$\text{fitness} = 1 - \frac{\text{custo\_total}}{\text{custo\_máximo\_possível}} \tag{3.3}$$

Definição do Custo: O custo representa o número de operações necessárias para fazer o modelo aceitar um trace do log. Durante a mineração, o custo inclui: (1) *movimentos no modelo* - transições silenciosas que precisam ser executadas, (2) *movimentos no log* - eventos que precisam ser ignorados, e (3) *movimentos síncronos* - eventos que coincidem perfeitamente (custo zero). Algoritmos de mineração usam esse custo para decidir quando parar de dividir o log e qual divisão produz o melhor modelo.

No contexto de alinhamentos, para cada trace individual:

$$\text{fitness\_trace}_i = 1 - \frac{\text{edit\_distance}_i}{\text{max\_length}_i} \tag{3.4}$$

O fitness global do log é calculado como:

$$\text{fitness\_log} = \frac{1}{n} \sum_{i=1}^{n} \text{fitness\_trace}_i \tag{3.5}$$

onde n é o número total de traces no log. O PM4Py implementa automaticamente esses cálculos utilizando algoritmos de alinhamento otimizados.

Interpretação:

- ≥ 0.9: modelo explica quase todo o log
- $0.7 - 0.9$: modelo explica maior parte
- < 0.7: modelo inadequado

3.4.7.2 Precision (0-1, maior é melhor)

Definição: quão preciso é o modelo, evitando comportamento extra não observado.

Método: token-based conformance.

Cálculo: A precision mede quantos comportamentos extras (não observados) o modelo permite. Utiliza a fórmula ETC (Escaping Arcs):

$$\text{precision} = 1 - \frac{\text{arcos\_escapando}}{\text{arcos\_totais}} \tag{3.6}$$

onde os arcos escapando representam transições que podem ser executadas pelo modelo mas não foram observadas no log original. O PM4Py calcula automaticamente essa métrica através de algoritmos de conformance checking que identificam comportamentos não observados.

Interpretação:

- ≥ 0.8: modelo preciso
- < 0.5: modelo muito generalista

3.4.7.3 Simplicity (0-1, maior é melhor)

Definição: simplicidade estrutural do modelo.

Método: razão entre arcos e nós na rede de Petri.

Cálculo: A simplicidade mede a complexidade topológica da Rede de Petri através da fórmula estrutural:

$$\text{simplicity} = \frac{1}{1 + \frac{|\text{arcos}|}{|\text{nós}|}} \tag{3.7}$$

onde $|\text{arcos}|$ representa o número total de arcos (transições) e $|\text{nós}|$ representa o número total de nós (places) na rede de Petri. O PM4Py calcula automaticamente essa métrica analisando a estrutura topológica do modelo descoberto.

Interpretação:

- ≥ 0.7: modelo simples
- < 0.4: modelo complexo

3.4.8 Saída da Etapa

Um objeto `ProcessModel` contendo:

- Rede de Petri (net, im, fm)
- Dicionário de estatísticas de atividades (`ActivityStatistics`)
- Métricas globais (arrival_rate, dispersion_rate, median_duration)
- Métricas de qualidade (fitness, precision, simplicity)
- Mapeamento de recursos (opcional)
- Perfil do log original (`LogProfile`)

3.5 Etapa 3: Simulação de Logs Sintéticos

3.5.1 Diagrama de Componente

A Figura 5 apresenta o diagrama de componente da Etapa 3, ilustrando o processo de simulação de logs sintéticos baseado em eventos discretos.

O componente central `LogSimulator` recebe como entradas o objeto `ProcessModel` contendo a Rede de Petri e todos os parâmetros estatísticos extraídos na Etapa 2, e um objeto `SimulationConfig` que especifica os parâmetros da simulação: `num_cases` (padrão 100) para número de casos a serem gerados, `arrival_rate` (extraído do modelo) para taxa de chegada de novos casos, `activity_durations` (extraídos do modelo) contendo as distribuições estatísticas de duração de cada atividade, e `random_seed` (padrão 42) para garantir reprodutibilidade dos experimentos.

O processamento interno implementa um simulador de eventos discretos utilizando a biblioteca SimPy, organizado em quatro componentes principais: (1) gerador de chegadas, que cria novos casos seguindo uma distribuição exponencial baseada na `arrival_rate` observada; (2) executor de casos, que para cada caso individual percorre a Rede de Petri respeitando sua semântica formal, identificando transições habilitadas e escolhendo uma aleatoriamente para disparo; (3) gerador de eventos, que para cada transição disparada cria um evento no log sintético com `case_id`, `activity`, `timestamp` (tempo atual da simulação) e `resource` (selecionado aleatoriamente do pool de recursos disponíveis para aquela atividade); (4) controlador de duração, que para cada atividade amostra uma duração da distribuição estatística ajustada (Normal, Log-Normal ou Exponencial) e avança o relógio da simulação pelo tempo correspondente. A simulação continua até que todos os casos alcancem a marcação final da Rede de Petri ou atinjam o limite de segurança (`max_trace_length=1000`) para proteção contra loops infinitos.

A saída produzida consiste em um objeto `SimulationResult` contendo: caminho para o arquivo CSV intermediário com os eventos gerados em formato tabular (`case_id`, `activity`, `time:timestamp`, `resource`); caminho para o arquivo XES final convertido seguindo o padrão IEEE 1849-2016, incluindo classificador de atividade para compatibilidade com ferramentas como ProM e Disco; estatísticas da simulação incluindo `num_cases_generated`, `num_events_generated`, `simulation_time_seconds` e `generation_timestamp`. As bibliotecas SimPy são utilizadas para implementação da lógica de eventos discretos (processos, timeouts, environment), enquanto PM4Py é empregada para conversão do formato intermediário CSV para o formato XES padronizado.

Figura 5 - LogSimulator - Simulação Computacional

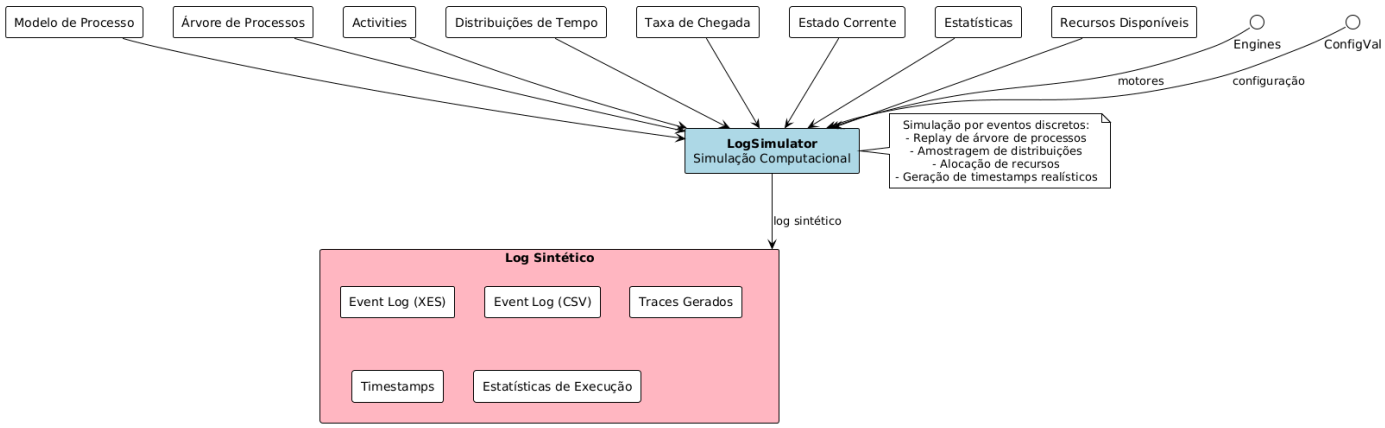

Figura 5 – Fase 3: Simulação de Logs Sintéticos. Fonte: Autor (2025)

3.5.2 Objetivo

Gerar novos casos de processo que sigam o modelo descoberto e as distribuições estatísticas extraídas, preservando características estruturais e temporais do processo original.

3.5.3 Paradigma de Simulação

3.5.3.1 Abordagem Escolhida

O sistema utiliza **Discrete Event Simulation (DES)** implementada com a biblioteca SimPy.

3.5.3.2 Justificativa

DES é adequado porque:

- Processos de negócio são inerentemente discretos (atividades têm início e fim definidos)
- Eventos ocorrem em pontos específicos do tempo

- Estado do sistema muda apenas em eventos
- SimPy oferece abstrações adequadas (processos, timeouts, recursos)

3.5.4 Configuração da Simulação

A Tabela 2 lista os parâmetros principais:

Tabela 2 – Parâmetros de configuração da simulação

Parâmetro	**Valor Padrão**	**Justificativa**
`num_cases`	100	Suficiente para análise estatística sem overhead
`arrival_rate`	Do modelo	Preserva taxa de chegada original
`activity_durations`	Do modelo	Preserva durações originais
`random_seed`	42	Reprodutibilidade dos experimentos
`max_trace_length`	1000	Limite de segurança contra loops

Todos os parâmetros podem ser sobrescritos para simulação de cenários alternativos (por exemplo, "E se reduzirmos durações em 50%?").

3.5.5 Geração de Casos

3.5.5.1 Processo de Chegadas

O sistema implementa um gerador SimPy que cria casos sequencialmente:

```
Para cada caso i de 1 até num_cases:
    1. Aguardar intervalo de chegada (arrival_rate)
    2. Iniciar processo paralelo para simular caso i
```

Este padrão permite múltiplos casos ativos simultaneamente, representando carga de trabalho real com concorrência.

3.5.6 Simulação Individual de Casos

O sistema implementa algoritmo baseado na semântica formal de Redes de Petri (PETERSON, 1981):

Algorithm 1 Simulação de um caso individual

Entrada: Rede de Petri (net, im, fm)
$marking \leftarrow im$ ▷ Marcação inicial
while $marking \neq fm$ **do** ▷ Até marcação final
    $T_{enabled} \leftarrow$ transições habilitadas em $marking$
    if $T_{enabled} = \emptyset$ **then**
        break ▷ Marcação final ou deadlock
    end if
    $t \leftarrow$ escolhe aleatoriamente de $T_{enabled}$
    if t não é transição silenciosa **then**
        $timestamp \leftarrow$ tempo_atual_simulação
        $recurso \leftarrow$ escolhe aleatório de recursos(t.atividade)
        Registrar evento($caso\_id, t.atividade, timestamp, recurso$)
        Aguardar duração(t.atividade) segundos
    end if
    $marking \leftarrow$ executa($t, marking$) ▷ Atualizar marcação
    if comprimento_trace $> max\_trace\_length$ **then**
        break ▷ Limite de segurança
    end if
end while

3.5.7 Geração dos Logs de Saída

3.5.7.1 Formato Intermediário (CSV)

```
case_id,activity,time:timestamp,resource
Case 1,Register Request,2024-10-13 10:00:00,John
Case 1,Examine,2024-10-13 10:02:30,Mary
...
```

3.5.7.2 Conversão para XES

Processo automatizado:

1. Leitura do CSV com Pandas
2. Renomeação de colunas para padrão XES IEEE
3. Formatação via `pm4py.format_dataframe()`
4. Conversão para estrutura de log PM4Py
5. Exportação XES via `pm4py.write_xes()`
6. Inserção manual de classificador (workaround limitação PM4Py)

O classificador XES inserido:

```
<classifier name="Activity" keys="concept:name"/>
```

É necessário para compatibilidade com ferramentas como ProM e Disco.

3.5.8 Reprodutibilidade e Determinismo

Um requisito fundamental da metodologia científica é a **reprodutibilidade**: a capacidade de outros pesquisadores replicarem exatamente os mesmos resultados a partir das mesmas entradas. A simulação de eventos discretos, por sua natureza estocástica, introduz aleatoriedade através de:

- Amostragem de durações de atividades de distribuições estatísticas
- Seleção aleatória de recursos do pool disponível
- Escolha de transições habilitadas em marcações com múltiplas opções
- Geração de tempos de chegada seguindo distribuição exponencial

Para garantir reprodutibilidade total, o sistema implementa as seguintes estratégias:

3.5.8.1 Geração de Números Aleatórios Baseada em Seed

O sistema utiliza um **seed fixo** (valor padrão: 42) para inicialização do gerador de números pseudoaleatórios. Isto garante que, para o mesmo seed, a mesma sequência de números aleatórios seja gerada, resultando em logs sintéticos idênticos:

```
config = SimulationConfig(
    num_cases=100,
    random_seed=42  # Reprodutibilidade total
)
```

O seed é propagado para:

- `numpy.random.seed()`: amostragem de distribuições
- `random.seed()`: seleção de recursos e transições
- SimPy environment: sequenciamento de eventos

3.5.8.2 Versionamento de Dependências

O arquivo `requirements.txt` fixa versões exatas de todas as bibliotecas:

```
pm4py==2.2.22
simpy==4.1.1
scipy==1.13.0
pandas==2.2.2
numpy==1.26.4
```

Isto previne comportamento divergente causado por atualizações de bibliotecas que possam alterar implementações de algoritmos.

3.5.8.3 Logging Completo de Configurações

Cada execução registra automaticamente:

- Parâmetros de configuração utilizados
- Seed do gerador de números aleatórios
- Versões das bibliotecas carregadas
- Hash SHA256 do arquivo de entrada
- Timestamp de execução

3.5.8.4 Determinismo na Ordem de Execução

Quando múltiplas transições estão habilitadas simultaneamente em uma marcação, a seleção é determinística dado o mesmo seed, evitando não-determinismo causado por ordenação de estruturas de dados.

3.5.8.5 Benefícios da Reprodutibilidade

1. **Validação Científica**: Outros pesquisadores podem verificar os resultados apresentados
2. **Debugging Determinístico**: Bugs podem ser reproduzidos exatamente para diagnóstico
3. **Comparação Justa**: Experimentos com diferentes parâmetros partem da mesma base aleatória
4. **Auditoria**: Resultados podem ser rastreados para suas configurações exatas
5. **Evolução Controlada**: Mudanças no código podem ser testadas contra baseline reproduzível

Para produzir resultados diferentes (por exemplo, para análise de sensibilidade), basta alterar o seed:

```
# Executar 10 replicações independentes
for seed in range(10):
    config = SimulationConfig(random_seed=seed)
    result = simulator.simulate(model, config)
```

3.5.9 Saída da Etapa

Um objeto `SimulationResult` contendo:

- Caminhos dos arquivos (CSV e XES)
- Número de casos e eventos gerados
- Tempo de execução da simulação
- Timestamp de geração
- Seed utilizado (para reprodutibilidade)
- Configuração completa da simulação

3.6 Etapa 4: Validação de Qualidade

3.6.1 Diagrama de Componente

A Figura 6 apresenta o diagrama de componente da Etapa 4, detalhando o processo de validação de qualidade dos logs sintéticos gerados.

O componente central `LogValidator` recebe como entradas dois arquivos XES: o arquivo `original_xes` contendo o log de eventos real e o arquivo `simulated_xes` contendo o log sintético gerado na Etapa 3.

O processamento interno implementa uma validação multidimensional que compara diferentes perspectivas dos logs: (1) **Análise Estrutural**, onde calcula-se a similaridade dos traces usando distância de edição; (2) **Análise de Frequência**, que compara as distribuições de atividades e variantes; (3) **Análise Temporal**, que mede a similaridade das distribuições de duração das atividades e aplica o teste de Kolmogorov-Smirnov para validação estatística.

A saída produzida consiste em um objeto estruturado `ValidationResult` contendo as métricas agregadas de todas as análises, incluindo percentual de similaridade, p-valor e

estatísticas detalhadas. A biblioteca PM4Py é utilizada para os cálculos de alinhamento e extração de características dos logs.

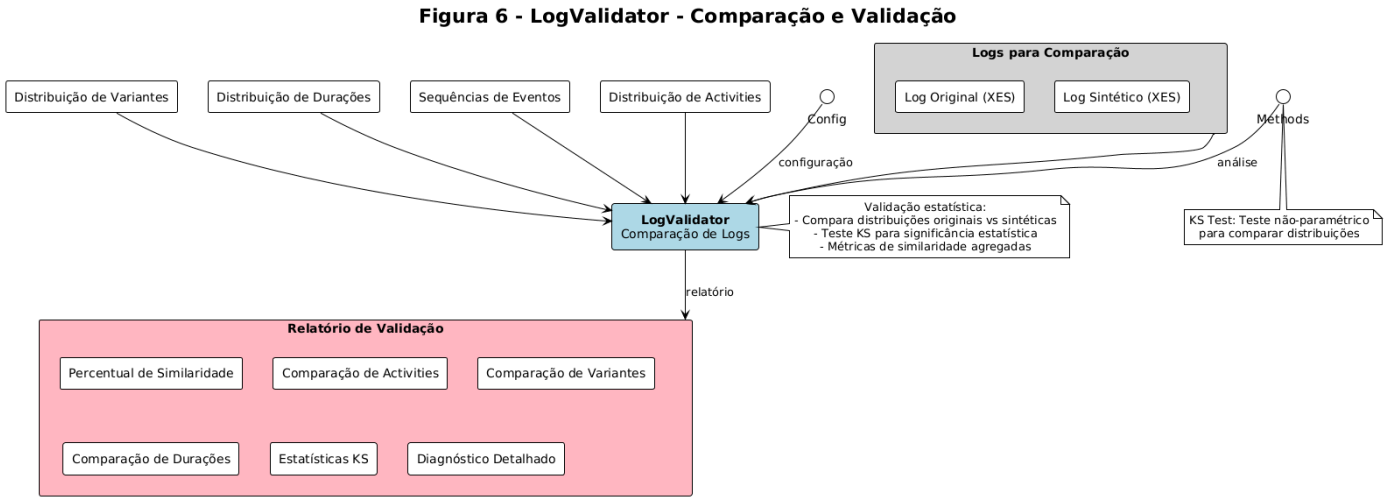

Figura 6 – Fase 4: Validação de Qualidade. Fonte: Autor (2025)

3.6.2 Objetivo

Avaliar quão similares são os logs original e sintético, validando a qualidade da geração e a fidelidade do modelo. A validação é multidimensional, comparando os logs sob perspectivas estrutural, de frequência e temporal.

3.6.3 Métricas de Validação

O sistema calcula um conjunto de métricas para fornecer uma visão holística da similaridade entre os logs.

3.6.3.1 Similaridade Estrutural (Fitness de Alinhamento)

A similaridade estrutural é calculada usando alinhamentos baseados em **Distância de Edição** (*Edit Distance*) entre os traces. Para cada par de traces (um original, um sintético), calcula-se o número mínimo de operações (inserção, deleção, substituição) para transformar um no outro. A partir do custo do alinhamento, deriva-se uma métrica de *fitness* normalizada:

$$fitness = 1 - \frac{\text{custo\_alinhamento}}{\text{custo\_máximo}} \tag{3.8}$$

Onde custo_máximo é o custo do pior caso (deletar todos os eventos de um trace e inserir todos do outro). O *fitness* médio de todos os alinhamentos resulta em um percentual de similaridade estrutural.

3.6.3.2 Similaridade de Distribuição de Atividades

Esta métrica compara a frequência relativa de cada atividade entre o log original e o sintético. Utiliza-se a distância de Jaccard ou similar para quantificar a sobreposição das distribuições de frequência. Um valor próximo de 1.0 indica que as atividades mais comuns no processo real também são as mais comuns no processo simulado.

3.6.3.3 Similaridade de Distribuição de Variantes

Compara a distribuição das variantes (sequências de atividades únicas) entre os dois logs. Esta métrica avalia se o modelo simulado consegue reproduzir os caminhos mais frequentes do processo real com uma frequência parecida. Assim como na métrica de atividades, um valor próximo de 1.0 é desejável.

3.6.3.4 Similaridade de Distribuição de Durações

Avalia se as durações das atividades no log sintético seguem uma distribuição estatística similar às do log original. A comparação é feita utilizando a distância de Wasserstein (Earth Mover's Distance) ou outra métrica de distância entre distribuições, que mede o "esforço"necessário para transformar uma distribuição na outra.

3.6.3.5 Validação Estatística (Teste de Kolmogorov-Smirnov)

Para uma validação mais rigorosa das durações, o sistema aplica o teste de Kolmogorov-Smirnov (K-S) de duas amostras. Este teste não paramétrico avalia a hipótese nula de que as amostras de duração (original e sintética) foram extraídas da mesma distribuição. O resultado é um **p-valor**:

- **p-valor > 0.05**: Não há evidência estatística para rejeitar a hipótese nula. As distribuições são consideradas estatisticamente similares.
- **p-valor $\leq$ 0.05**: A hipótese nula é rejeitada. As distribuições são consideradas estatisticamente diferentes.

Um p-valor alto indica que a simulação reproduz fielmente a variabilidade temporal do processo original.

3.6.4 Agregação e Interpretação de Resultados

As métricas são agregadas em um placar de validação. A Tabela 3 apresenta os thresholds para interpretação.

Tabela 3 – Interpretação de métricas de validação

Métrica	**Interpretação**
Similaridade Estrutural	$\geq 80\%$ (Boa), $\geq 90\%$ (Excelente)
Similaridade (Atividades)	≥ 0.8 (Boa)
Similaridade (Variantes)	≥ 0.7 (Boa)
Similaridade (Durações)	≥ 0.8 (Boa)
KS-Test p-valor	> 0.05 (Similaridade Estatística)

3.6.5 Saída da Etapa

Um objeto `ValidationResult` contendo:

- Percentual de similaridade estrutural (baseado em fitness médio)
- Similaridade de distribuição de atividades, variantes e durações
- P-valor do teste de Kolmogorov-Smirnov
- Detalhes estatísticos (médias, mínimos, máximos de fitness e custo)

3.7 Componente Independente: Cálculo de Métricas ORE

3.7.1 Posicionamento no Sistema

O **ORECalculator** é um componente independente e opcional do sistema, especializado em calcular métricas de eficiência operacional para processos hospitalares, especificamente para salas cirúrgicas. Ao contrário dos quatro componentes anteriores que formam o pipeline sequencial obrigatório (LogAnalyzer $\rightarrow$ ProcessMiner $\rightarrow$ LogSimulator $\rightarrow$ LogValidator), o ORECalculator opera de forma **paralela e independente**, podendo ser invocado diretamente sobre o log original sem necessidade de executar as etapas de mineração ou simulação.

3.7.2 Arquitetura do Componente

Figura 7 – Arquitetura do componente ORECalculator. Fonte: Autor (2025)

3.7.3 Justificativa da Separação Arquitetural

A decisão de implementar o ORECalculator como componente independente fundamenta-se em três razões principais:

1. **Especificidade de Domínio**: As métricas ORE são específicas para processos hospitalares (salas cirúrgicas), enquanto os componentes do pipeline principal são **domain-agnostic**,

funcionando para qualquer tipo de processo de negócio (manufatura, finanças, logística, etc.).

2. **Independência Funcional**: O cálculo de ORE não depende da execução do pipeline de simulação. Um analista hospitalar pode querer apenas calcular indicadores de eficiência do log real, sem interesse em gerar logs sintéticos.

3. **Extensibilidade**: Esta arquitetura permite adicionar outros calculadores específicos de domínio no futuro (por exemplo, métricas financeiras, KPIs de manufatura) sem acoplar ao pipeline principal.

3.7.4 Framework ORE (Operating Room Effectiveness)

O sistema implementa o framework ORE proposto por Souza et al. (SOUZA; VACCARO; LIMA, 2020), que decompõe a efetividade operacional de salas cirúrgicas em três dimensões multiplicativas:

$$\text{ORE} = \text{Disponibilidade} \times \text{Desempenho} \times \text{Qualidade} \tag{3.9}$$

3.7.4.1 Disponibilidade (A)

Representa a razão entre o tempo total programado e o tempo total disponível:

$$A = \frac{\text{TTS}}{\text{TTA}} \tag{3.10}$$

Interpretação: Que percentual do tempo disponível foi efetivamente agendado para cirurgias?

Categorias de perdas que impactam A:

- Falha de equipamento
- Tempo de setup (preparação da sala)
- Não agendamento (slots vagos)

3.7.4.2 Desempenho (P)

Representa a razão entre o tempo total utilizado e o tempo total programado:

$$P = \frac{\text{TTU}}{\text{TTS}} \tag{3.11}$$

Interpretação: Que percentual do tempo agendado foi efetivamente utilizado?

Categorias de perdas que impactam P:

- Pequenas paradas (atrasos operacionais)
- Variação no tempo cirúrgico
- Cancelamentos

3.7.4.3 Qualidade (Q)

Representa a razão entre o tempo total de valor agregado e o tempo total utilizado:

$$Q = \frac{\text{TTAV}}{\text{TTU}} \tag{3.12}$$

Interpretação: Que percentual do tempo utilizado agregou valor real (sem retrabalho)?

Categorias de perdas que impactam Q:

- Reintervenções (cirurgias repetidas)

3.7.5 Suporte a XES Enriquecido

O componente possui capacidade de processar dois tipos de logs XES:

3.7.5.1 XES Padrão

Quando o log contém apenas atributos padrão (activity, timestamp, resource), o sistema **estima** as métricas ORE baseando-se em heurísticas:

- Duração planejada estimada pela mediana das durações observadas
- Cancelamentos inferidos de casos incompletos
- Perdas estimadas por valores padrão da literatura

3.7.5.2 XES Enriquecido

Quando o log contém atributos customizados específicos do domínio hospitalar, o sistema calcula métricas **exatas**:

Atributos enriquecidos suportados:

- `surgery:status`: status da cirurgia (Realizada, Cancelada, Pendente)
- `surgery:actual_duration`: duração real da cirurgia em minutos
- `surgery:cancellation_reason`: motivo do cancelamento (Paciente, Equipamento, Recurso)

- `surgery:reintervention`: flag booleana indicando reintervenção

Exemplo de trace enriquecido:

```
<trace>
  <string key="concept:name" value="Case001"/>
  <event>
    <string key="concept:name" value="Surgery"/>
    <date key="time:timestamp" value="2024-10-13T08:00:00"/>
    <string key="surgery:status" value="Realizada"/>
    <int key="surgery:actual_duration" value="125"/>
    <boolean key="surgery:reintervention" value="false"/>
  </event>
</trace>
```

3.7.6 Decomposição Granular de Perdas

O sistema categoriza perdas operacionais em sete categorias conforme o framework ORE:

Tabela 4 – Categorias de perdas operacionais no framework ORE

Categoria de Perda	Descrição
Falha de Equipamento	Tempo perdido por quebra ou indisponibilidade de equipamentos
Setup	Tempo de preparação da sala entre cirurgias (padrão: 15 min/caso)
Não Agendamento	Slots disponíveis não utilizados (TTA - TTS)
Pequenas Paradas	Interrupções breves durante procedimentos
Variação no Tempo Cirúrgico	Diferença entre tempo planejado e tempo real
Cancelamentos	Cirurgias agendadas mas não realizadas
Reintervenções	Repetição de procedimentos (retrabalho)

3.7.7 Integração com o Pipeline

Embora independente, o ORECalculator pode ser invocado através da fachada `ProcessMiningPipeline`:

```
# Uso independente
ore_calc = ORECalculator(daily_hours=11.5, setup_time_minutes=15)
metrics = ore_calc.calculate_from_log("surgical_log.xes")
```

```
# Uso através do pipeline
pipeline = ProcessMiningPipeline()
ore_metrics = pipeline.calculate_ore("surgical_log.xes")

# Uso integrado com pipeline completo
results = pipeline.run_full_pipeline("surgical_log.xes")
# results['ore_metrics'] contém OREMetrics (se aplicável)
```

3.7.8 Saída do Componente

Um objeto `OREMetrics` contendo:

- **Indicadores**: ore, availability, performance, quality (0-1)
- **Componentes temporais**: TTA, TTS, TTU, TTAV (horas)
- **Decomposição de perdas**: Dict com 7 categorias (horas)
- **Estatísticas de cancelamento**: taxa, motivos, contagens
- **Cobertura de dados**: percentual de eventos com cada atributo enriquecido

3.8 Interfaces de Acesso ao Sistema

O Sim2Log-Core oferece duas interfaces complementares para diferentes perfis de usuários, implementando uma estratégia de **multi-interface** que amplia a acessibilidade do sistema.

3.8.1 Interface Programática (API Python)

3.8.1.1 Público-Alvo

Pesquisadores, cientistas de dados e desenvolvedores que necessitam integrar o sistema em scripts, notebooks Jupyter ou pipelines de análise automatizados.

3.8.1.2 Características

Alto nível (via Facade):

```
from pipeline import ProcessMiningPipeline, quick_analysis

# Uso simplificado
results = quick_analysis("log.xes")
```

```
# Uso com controle
pipeline = ProcessMiningPipeline(verbose=True)
profile = pipeline.analyze_log("log.xes")
model = pipeline.mine_process("log.xes", variant_filter=0.8)
sim_result = pipeline.simulate(model, num_cases=100)
validation = pipeline.validate("original.xes", "synthetic.xes")
```

Nível de componente (uso independente):

```
from log_analyzer import LogAnalyzer
from process_mining import ProcessMiner
from simulation import LogSimulator, SimulationConfig

# Uso granular de componentes individuais
analyzer = LogAnalyzer()
profile = analyzer.analyze("log.xes")

miner = ProcessMiner()
model = miner.mine_process("log.xes")

config = SimulationConfig(num_cases=200, random_seed=123)
simulator = LogSimulator(config)
result = simulator.simulate(model)
```

3.8.1.3 Vantagens

- **Automação**: Processamento em lote de múltiplos logs
- **Integração**: Incorporação em workflows existentes
- **Flexibilidade**: Acesso granular a componentes individuais
- **Reprodutibilidade**: Scripts versionados garantem replicação exata
- **Programabilidade**: Controle total via código Python

3.8.2 Interface Web Interativa (Streamlit Dashboard)

3.8.2.1 Público-Alvo

Analistas de negócios, profissionais de saúde, gestores hospitalares e usuários sem conhecimento de programação que necessitam de análise exploratória e validação de processos.

3.8.2.2 Funcionalidades

1. **Upload de Arquivos**: Drag-and-drop de arquivos XES
2. **Execução Dual-Mode**:
    - *Run Everything*: Pipeline completo com um clique
    - *Step-by-Step*: Controle individual de cada etapa
3. **Visualizações Interativas**:
    - Diagrama da Rede de Petri (via Graphviz)
    - Gráficos de barras de frequência de atividades
    - Distribuição de variantes do processo
    - Comparação visual de métricas de qualidade
4. **Download de Resultados**:
    - Logs sintéticos (XES e CSV)
    - Modelo de processo (PNML)
    - Relatórios de validação (JSON)
    - Métricas ORE (JSON)
5. **Feedback em Tempo Real**: Barras de progresso e mensagens de status
6. **Tratamento de Erros**: Mensagens descritivas para problemas comuns

3.8.2.3 Implementação Técnica

A interface é implementada usando Streamlit 1.28.0, que permite criar aplicações web interativas usando apenas Python, sem necessidade de conhecimento de HTML, CSS ou JavaScript. A comunicação com o backend ocorre através da instância `ProcessMiningPipeline`, mantendo a separação entre interface e lógica de negócio.

Execução:

```
streamlit run app/app.py
# Acesso em http://localhost:8501
```

3.8.2.4 Vantagens

- **Acessibilidade**: Usuários não-técnicos podem utilizar o sistema
- **Exploração Interativa**: Ajuste de parâmetros em tempo real

- **Visualização Imediata**: Resultados apresentados graficamente
- **Sem Instalação Cliente**: Acesso via navegador web
- **Compartilhamento Fácil**: Interface pode ser hospedada e compartilhada

3.8.3 Comparação entre Interfaces

Tabela 5 – Comparação entre as interfaces de acesso

Critério	**API Programática**	**Interface Web**
Público-alvo	Pesquisadores, desenvolvedores	Analistas, gestores
Conhecimento requer.	Python intermediário	Nenhum (uso intuitivo)
Controle	Total (nível de código)	Limitado a parâmetros expostos
Automação	Fácil (scripts, batch)	Difícil (interativa)
Visualização	Requer código adicional	Integrada
Reprodutibilidade	Excelente (scripts versionados)	Moderada (config manual)
Curva de aprendizado	Íngreme	Suave
Uso típico	Análise em lote, integração, pesquisa	Exploração, validação, demonstração

3.9 Parâmetros e Configurações

3.9.1 Tabela de Parâmetros Principais

A Tabela 6 apresenta todos os parâmetros configuráveis do sistema:

Tabela 6 – Parâmetros configuráveis do sistema

Parâmetro	**Padrão**	**Faixa**	**Impacto**
`variant_filter`	0.8	0.0-1.0	Filtragem de ruído na mineração
`num_cases`	100	1-∞	Tamanho do log sintético
`arrival_rate`	Auto	0.1-∞ min	Taxa de chegada de casos
`random_seed`	42	0-2^{32}-1	Reprodutibilidade
`max_trace_length`	1000	1-∞	Proteção contra loops
`verbose`	True	True/False	Saída de diagnóstico
`save_model_image`	None	Path/None	Visualização do modelo

3.10 Ferramentas e Tecnologias

3.10.1 Bibliotecas Principais

A Tabela 7 apresenta as tecnologias utilizadas.

Tabela 7 – Bibliotecas e ferramentas utilizadas

Biblioteca	**Versão**	**Justificativa**
PM4Py	2.2.22	Padrão de facto em Python para process mining, algoritmos state-of-the-art
SimPy	4.0.1	Leve, Pythônico, documentação excelente para DES
SciPy	1.13.0	Completa, bem testada, K-S test built-in
Pandas	2.2.2	Eficiente para manipulação de dados, operações vetorizadas
NumPy	1.26.4	Base do ecossistema científico Python
Streamlit	1.28.0	Interface web rápida e intuitiva
Graphviz	0.20.3	Visualização de Redes de Petri

3.10.2 Justificativa das Escolhas

A escolha das bibliotecas foi fundamentada em critérios técnicos específicos para cada funcionalidade do sistema. O PM4Py representa o padrão de facto em Python para process mining, oferecendo algoritmos state-of-the-art com documentação completa. O SimPy foi selecionado para simulação discreta de eventos por sua simplicidade e adequação perfeita para modelagem de sistemas discretos. O SciPy foi escolhido para análise estatística por sua robustez e confiabilidade.

Para manipulação de dados, o Pandas foi selecionado por sua eficiência em operações vetorizadas, essencial para processamento dos logs de eventos. O NumPy serve como base fundamental do ecossistema científico Python, proporcionando operações matemáticas otimizadas. O Streamlit foi escolhido para criar interfaces web rápidas e intuitivas, facilitando a interação com o sistema. O Graphviz foi selecionado especificamente para visualização de Redes de Petri, oferecendo capacidades gráficas adequadas para representação dos modelos de processo descobertos.

3.11 Pseudocódigo de Alto Nível

O algoritmo 2 apresenta o pipeline completo:

Algorithm 2 Pipeline completo de geração de logs sintéticos

function GENERATESYNTHETICLOG($original\_xes\_path$)
    // Etapa 1: Análise
    $analyzer \leftarrow$ LOGANALYZER
    $profile \leftarrow analyzer.analyze(original\_xes\_path)$

    // Etapa 2: Mineração
    $miner \leftarrow$ PROCESSMINER(verbose=True)
    $model \leftarrow miner.mine\_process(original\_xes\_path, variant\_filter = 0.8)$

    // Etapa 3: Simulação
    $config \leftarrow$ SIMULATIONCONFIG(num_cases=100, random_seed=42)
    $simulator \leftarrow$ LOGSIMULATOR($config$, verbose=True)
    $result \leftarrow simulator.simulate(model, output\_dir)$

    // Etapa 4: Validação
    $validator \leftarrow$ LOGVALIDATOR(verbose=True)
    $validation \leftarrow validator.validate(original\_xes\_path, result.xes\_path)$

    return $(result, validation)$
end function

4 Desenvolvimento

Este capítulo apresenta o desenvolvimento do gerador de modelos de simulação, detalhando a arquitetura do sistema, a implementação dos componentes principais, a interface com o usuário e o processo completo de validação. O desenvolvimento seguiu princípios de modularidade, separação de responsabilidades e generalização, buscando criar uma solução que pudesse ser aplicada a diferentes domínios sem necessidade de parametrização manual. As escolhas arquiteturais e de implementação foram norteadas pelos requisitos de automatização, robustez e reprodutibilidade estabelecidos nos objetivos do trabalho.

4.1 Estrutura do Projeto

O código-fonte do sistema está organizado em uma estrutura modular que separa claramente as responsabilidades e facilita a manutenção e extensão. A estrutura completa do projeto é apresentada a seguir:

```
main/
|-- venv/                      # Ambiente virtual Python
|-- bases/                     # Logs XES de entrada (gitignored)
|-- output/                    # Resultados gerados (gitignored)
|
|-- core/                      # Módulo principal
|   |-- __init__.py          # Exportações da API
|   |-- models.py            # Dataclasses
|   |-- log_analyzer.py      # Análise automática de logs
|   |-- process_mining.py    # Mineração de processos
|   |-- simulation.py        # Simulação de eventos discretos
|   |-- validation.py        # Validação de qualidade
|   |-- utils.py             # Funções auxiliares
|   |-- excel_to_xes.py      # Converte Excel para XES
|
|-- app/                       # Interface web Streamlit
|   |-- app.py               # Aplicação principal
|   |-- uploads/             # Arquivos XES carregados
|   |-- outputs/             # Resultados da interface
|
|-- test.py                    # Script de teste CLI
```

4.1.1 Organização dos Módulos

A estrutura modular do projeto separa claramente as responsabilidades:

- **core/**: Contém toda a lógica de negócio do sistema, incluindo análise de logs, mineração de processos, simulação e validação
- **app/**: Interface web desenvolvida com Streamlit para facilitar o uso do sistema
- **bases/** e **output/**: Diretórios para dados de entrada e saída, respectivamente (ignorados pelo Git)

4.2 Arquitetura do Sistema

4.2.1 Filosofia Arquitetural

O Sim2Log-Core foi projetado seguindo uma **arquitetura em camadas modular** com clara separação de responsabilidades. Esta abordagem arquitetural fundamenta-se em princípios de engenharia de software estabelecidos, incluindo Separação de Responsabilidades (*Separation of Concerns*), Princípio da Responsabilidade Única (*Single Responsibility Principle*) e Inversão de Dependências (*Dependency Inversion*). A arquitetura foi estruturada em cinco camadas hierárquicas, conforme ilustrado na Figura 8.

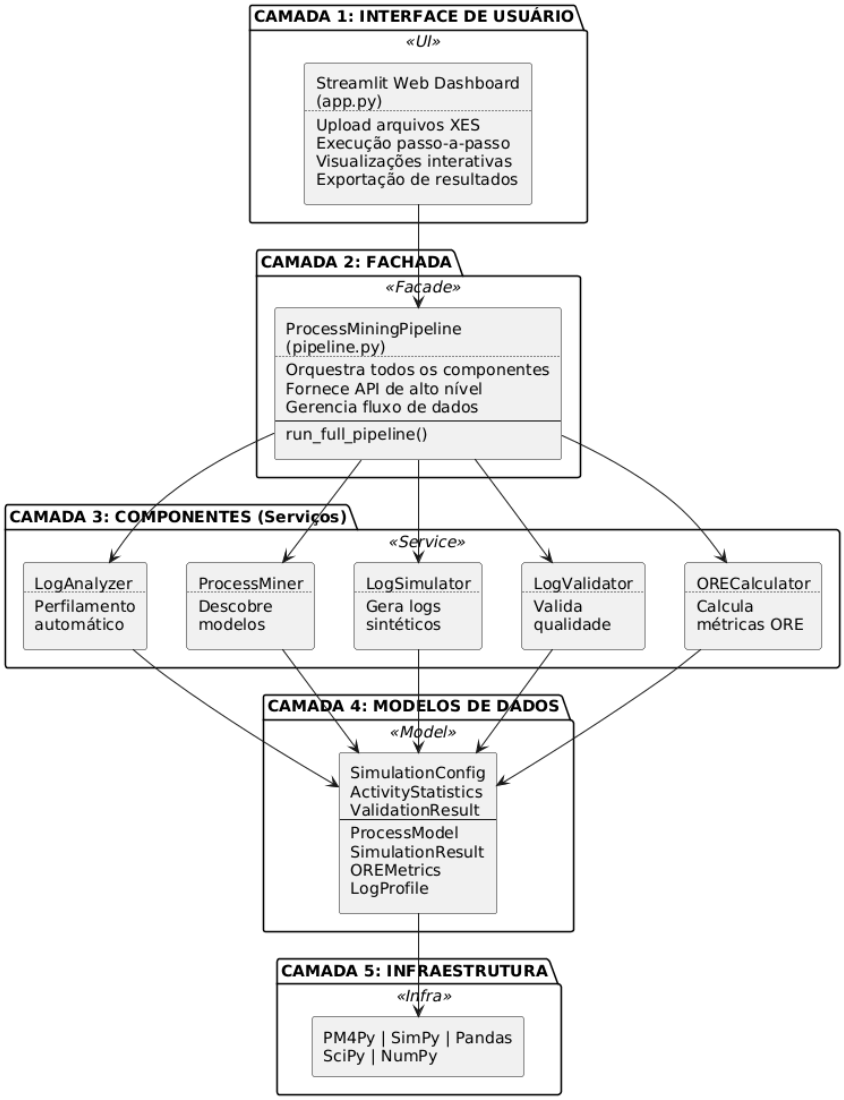

Figura 8 – Arquitetura em camadas do Sim2Log-Core. Fonte: Autor (2025)

4.2.1.1 Camada 1: Interface de Usuário

Esta camada fornece o ponto de acesso para usuários finais através de um dashboard web interativo implementado com Streamlit. A interface permite operações de upload, configuração, execução e visualização de resultados sem necessidade de programação.

4.2.1.2 Camada 2: Fachada

Implementa o padrão de projeto Facade através da classe `ProcessMiningPipeline`, que fornece uma interface simplificada e unificada para o subsistema complexo. Esta camada oculta a complexidade da coordenação entre os cinco componentes independentes, oferecendo métodos de alto nível como `run_full_pipeline()` e `quick_analysis()`.

4.2.1.3 Camada 3: Componentes

Cada componente é uma unidade independente e testável que implementa uma responsabilidade específica do sistema. Esta arquitetura baseada em componentes permite:

- **Modularidade**: componentes podem ser usados independentemente
- **Testabilidade**: cada componente possui testes isolados
- **Manutenibilidade**: mudanças em um componente não afetam outros
- **Extensibilidade**: novos componentes podem ser adicionados sem modificar existentes

4.2.1.4 Camada 4: Modelos de Dados

Define estruturas de dados fortemente tipadas usando Python dataclasses, garantindo type safety e auto-documentação. Estas classes servem como contratos de interface entre os componentes.

4.2.1.5 Camada 5: Infraestrutura

Bibliotecas de terceiros que fornecem funcionalidades fundamentais de mineração de processos, simulação, análise estatística e manipulação de dados.

4.2.2 Padrões de Projeto Utilizados

4.2.2.1 Padrão Facade (Fachada)

A classe `ProcessMiningPipeline` implementa o padrão Facade (GAMMA et al., 1994), fornecendo uma interface unificada e simplificada para o subsistema complexo de cinco componentes independentes.

Problema: A utilização direta dos componentes individuais requer conhecimento detalhado de suas interfaces, dependências e ordem de execução, aumentando a complexidade para o usuário.

Solução: O facade encapsula a complexidade da orquestração, oferecendo métodos simples:

```
# Interface simplificada
pipeline = ProcessMiningPipeline()
results = pipeline.run_full_pipeline("log.xes")

# Em vez de:
# analyzer = LogAnalyzer()
```

```
# profile = analyzer.analyze("log.xes")
# miner = ProcessMiner()
# model = miner.mine_process("log.xes")
# simulator = LogSimulator(config)
# result = simulator.simulate(model)
# validator = LogValidator()
# validation = validator.validate("original.xes", "synthetic.xes")
```

Benefícios:

- Redução da complexidade para o código cliente
- Desacoplamento entre cliente e implementação interna
- Facilita a orquestração do workflow completo
- Permite evolução interna sem quebrar código cliente

4.2.2.2 Arquitetura Baseada em Componentes

Cada componente (`LogAnalyzer`, `ProcessMiner`, `LogSimulator`, `LogValidator`, `ORECalculator`) é projetado para ser independente e reutilizável:

```
# Cada componente pode ser usado isoladamente
analyzer = LogAnalyzer()
profile = analyzer.analyze("log.xes")

miner = ProcessMiner()
model = miner.mine_process("log.xes")

simulator = LogSimulator(config)
result = simulator.simulate(model)
```

Benefícios:

- Modularidade e coesão alta
- Acoplamento baixo entre componentes
- Testabilidade independente
- Facilita manutenção e evolução

4.2.2.3 Data Classes para Type Safety

O sistema utiliza Python dataclasses com type hints para garantir segurança de tipos e auto-documentação:

```
@dataclass
class ProcessModel:
    petri_net: object
    initial_marking: object
    final_marking: object
    activities: Dict[str, ActivityStatistics]
    arrival_rate: float
    median_case_duration: float
    quality_metrics: Dict[str, float]
```

Benefícios:

- Type hints para suporte de IDEs
- Código auto-documentado
- Validação em tempo de desenvolvimento
- Serialização facilitada

4.2.2.4 Padrão Strategy para Ajuste de Distribuições

O ajuste de distribuições estatísticas implementa o padrão Strategy, testando múltiplas estratégias (Normal, Log-Normal, Exponencial) e selecionando a melhor através do teste de Kolmogorov-Smirnov:

```
# Tenta Normal → Log-Normal → Exponencial
# Seleciona melhor via teste KS (maior p-valor)
best_fit = fitter.fit(durations)
```

4.2.3 Arquitetura de Classes

O sistema é composto por 12 classes principais organizadas em três grupos: (1) a classe orquestradora ProcessMiningPipeline, (2) cinco componentes de serviço independentes, e (3) sete classes de modelos de dados. A Figura 9 apresenta uma visão geral simplificada das classes e seus relacionamentos.

Figura 9 – Visão geral da arquitetura de classes do Sim2Log-Core. Fonte: Autor (2025)

4.2.3.1 Classe Orquestradora: ProcessMiningPipeline

A classe ProcessMiningPipeline implementa o padrão Facade, fornecendo uma interface unificada para todo o sistema. A Figura 10 detalha seus atributos e métodos.

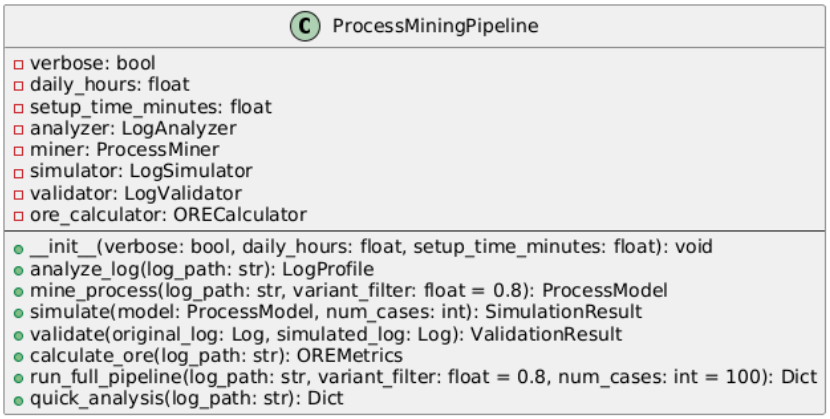

Figura 10 – Classe ProcessMiningPipeline com atributos e métodos detalhados. Fonte: Autor (2025)

4.2.3.2 Componentes de Serviço

Os cinco componentes principais (LogAnalyzer, ProcessMiner, LogSimulator, LogValidator, ORECalculator) são classes independentes, cada uma responsável por uma etapa específica do processamento. A Figura 11 apresenta a estrutura detalhada de cada componente.

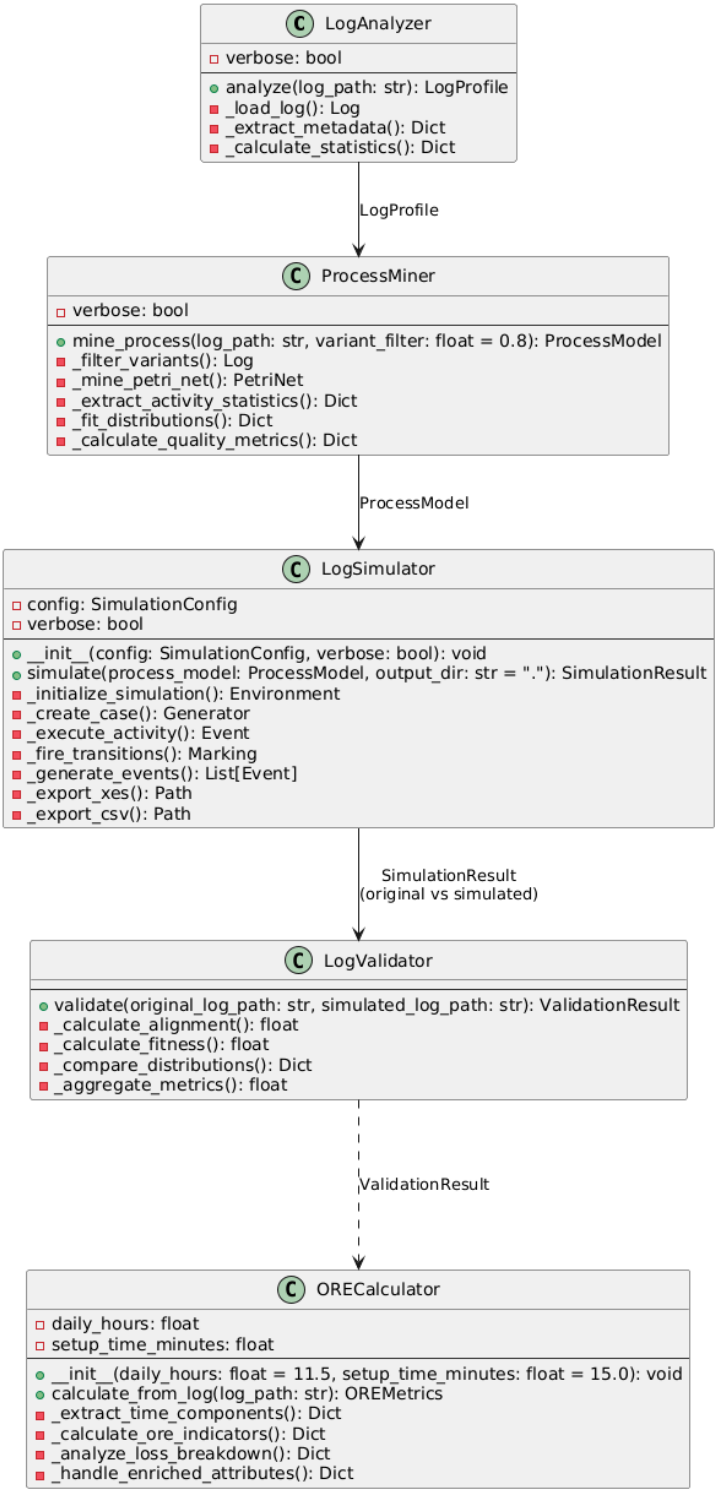

Figura 11 – Classes de componentes com métodos principais e LOC. Fonte: Autor (2025)

4.2.3.3 Modelos de Dados

As sete dataclasses (LogProfile, ProcessModel, ActivityStatistics, SimulationConfig, SimulationResult, ValidationResult, OREMetrics) definem estruturas de dados fortemente tipadas que fluem entre os componentes. A Figura 12 detalha a estrutura de cada modelo.

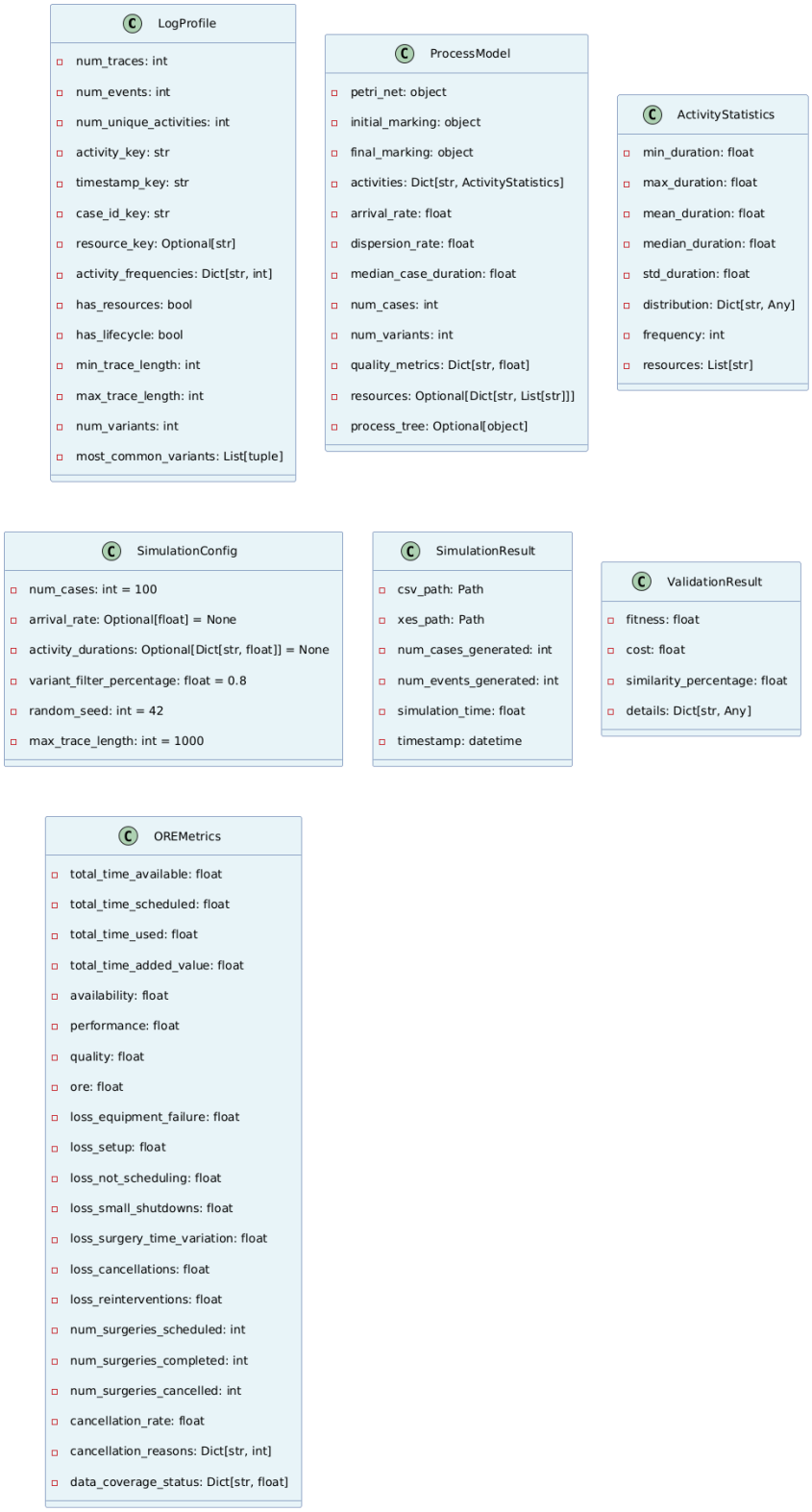

Figura 12 – Classes de modelos de dados com atributos principais. Fonte: Autor (2025)

4.2.4 Princípios de Design Seguidos

O desenvolvimento do Sim2Log-Core fundamentou-se em princípios estabelecidos de engenharia de software:

Tabela 8 – Princípios de design aplicados no sistema

Princípio	Implementação no Sistema
Separação de Responsabilidades	Cada módulo tem responsabilidade única e bem definida
DRY (Don't Repeat Yourself)	Funções comuns centralizadas no módulo utils.py
Aberto/Fechado	Fácil extensão sem modificar código existente (novos componentes podem ser adicionados)
Inversão de Dependências	Componentes dependem de abstrações (dataclasses) não de implementações concretas
SOLID	Responsabilidade única, injeção de dependências, segregação de interfaces
Fail-Fast	Validações no início de cada método, lançamento de exceções descritivas
Composition over Inheritance	Pipeline compõe componentes em vez de herança
Single Source of Truth	models.py define estruturas de dados usadas por todos
Reproducibility	Seed fixo, versionamento de dependências, logging completo
Graceful Degradation	Sistema funciona mesmo com dados parciais (ex: recursos opcionais, atributos enriquecidos opcionais)

4.3 Implementação dos Componentes Principais

Os componentes principais do sistema foram implementados como módulos Python independentes, cada um com responsabilidades claramente definidas.

4.3.1 Módulo de Análise de Logs

O módulo de análise (log_analyzer.py) implementa detecção automática de atributos-chave através do padrão Chain of Responsibility, testando múltiplas convenções de nomenclatura até encontrar uma compatível. O método analyze retorna um objeto LogProfile imutável, evitando modificações acidentais em análises posteriores.

4.3.2 Módulo de Mineração de Processos

O módulo de mineração (process_mining.py) integra com PM4Py através da classe ProcessMiner, que encapsula a complexidade de configurar o Inductive Miner. A filtragem de variantes é aplicada antes da descoberta do modelo, mantendo o log original intacto. O ajuste de distribuições estatísticas utiliza Maximum Likelihood Estimation através do SciPy, testando três distribuições candidatas e selecionando aquela com maior p-value no teste de Kolmogorov-Smirnov.

4.3.3 Módulo de Simulação

O módulo de simulação (simulation.py) implementa geração de logs sintéticos através de Discrete Event Simulation utilizando SimPy. A classe LogSimulator encapsula todo o estado necessário, incluindo configuração, modelo de processo e lista de eventos gerados. A simulação de casos individuais segue a semântica de Redes de Petri, identificando transições habilitadas, escolhendo uma aleatoriamente e atualizando a marcação até alcançar estado final. Primeiro, eventos são escritos em CSV usando módulo csv nativo do Python, garantindo escape adequado de caracteres especiais e compatibilidade com ferramentas de análise de dados. Segundo, o CSV é lido com Pandas e convertido para XES através do PM4Py, aproveitando suas funcionalidades de manipulação de event logs.

4.3.4 Módulo de Validação

O módulo de validação (validation.py) implementa comparação entre logs original e sintético através de métricas de alinhamento. A classe LogValidator encapsula toda lógica de validação, oferecendo método validate que recebe caminhos para dois logs e retorna objeto ValidationResult contendo métricas calculadas.

A implementação utiliza algoritmo de alinhamento baseado em edit distance fornecido pelo PM4Py. Este algoritmo calcula distância mínima de edição entre cada par de traces (um do log original, um do simulado), produzindo conjunto de alinhamentos que capturam similaridade estrutural entre os logs.

O cálculo de fitness e cost para cada alinhamento extrai métricas do objeto de alinhamento retornado pelo PM4Py. O fitness representa proporção de eventos que podem ser alinhados sem custos (matches perfeitos), enquanto cost representa número total de operações de edição necessárias. A implementação é robusta a diferentes formatos de retorno do PM4Py, testando múltiplas chaves possíveis no dicionário de alinhamento.

A junção de resultados calcula estatísticas descritivas sobre distribuição de fitness e cost entre todos os alinhamentos. Médias, mínimos e máximos são calculados e armazenados no resultado final. O percentual de similaridade é derivado diretamente do fitness médio multiplicado

por cem, oferecendo métrica intuitiva para usuários não técnicos.

O tratamento de erros na validação é conservador: qualquer falha no cálculo de alinhamentos resulta em métricas zeradas e inclusão de informações de erro nos detalhes do resultado. Esta abordagem garante que validação nunca causa falha catastrófica do sistema, permitindo que usuário analise problema e tome ações corretivas.

4.4 Implementação da Interface de Usuário

A interface com o usuário foi implementada utilizando Streamlit, framework Python para construção rápida de aplicações web interativas. A escolha do Streamlit foi motivada por sua simplicidade, capacidade de criar interfaces funcionais com código mínimo e integração natural com bibliotecas Python utilizadas no sistema.

4.4.1 Tecnologias Utilizadas

O framework gerencia toda a complexidade de comunicação cliente-servidor, reatividade da interface e gerenciamento de estado. O gerenciamento de estado persistente entre reexecuções utiliza st.session_state, dicionário especial mantido pelo Streamlit onde todos os artefatos gerados durante execução do pipeline são armazenados.

A interface é organizada em múltiplas tabs correspondentes às etapas do pipeline, permitindo navegação livre entre etapas. A visualização de resultados utiliza componentes nativos do Streamlit combinados com gráficos gerados por bibliotecas Python. Métricas são exibidas usando st.metric, gráficos de barras utilizam st.bar_chart e tabelas são renderizadas com st.dataframe.

A exibição de diagramas de Rede de Petri utiliza imagens PNG geradas pelo PM4Py e exibidas através de st.image. O upload de arquivos utiliza st.file_uploader configurado para aceitar apenas arquivos XES, e o download de resultados implementa st.download_button para cada arquivo gerado. O feedback ao usuário durante operações longas utiliza st.spinner com indicador de progresso animado. A figura a seguir ilustra a interface do usuário:

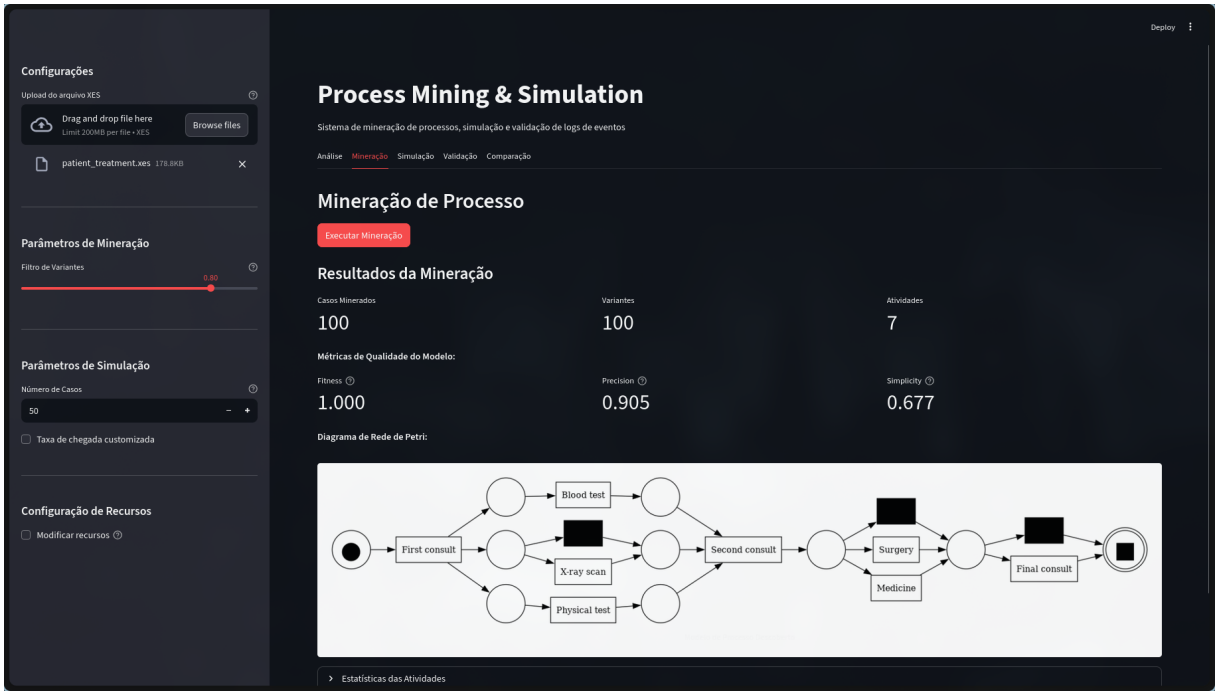

Figura 13 – Interface do usuário. Fonte: Autor (2025)

4.5 Testes e Validação

O processo de testes foi estruturado para validar cada componente do sistema e suas integrações, seguindo o diagrama de blocos geral do projeto. Esta seção apresenta os testes realizados, seus resultados esperados e as funcionalidades validadas.

4.5.1 Teste de Análise Automática de Logs

Descrição: Teste da funcionalidade de detecção automática de atributos-chave e coleta de estatísticas estruturais de logs XES.

Resultado Esperado: O sistema deve detectar corretamente os atributos case_id, activity_name e timestamp, extrair estatísticas como número de casos, atividades únicas, variantes e distribuição de durações.

Funcionalidade Validada: Etapa 1 do pipeline - Análise Automática de Logs, incluindo detecção de atributos-chave e coleta de estatísticas estruturais e temporais.

4.5.2 Teste de Mineração de Processos

Descrição: Teste da descoberta de modelo de processo através do Inductive Miner e ajuste de distribuições estatísticas.

Resultado Esperado: Geração de Rede de Petri válida, estatísticas de atividades com distribuições ajustadas (Normal, Log-Normal ou Exponencial) e métricas de qualidade (fitness > 0.7).

Funcionalidade Validada: Etapa 2 do pipeline - Mineração de Processos, incluindo filtragem de variantes, descoberta do modelo e extração de estatísticas temporais.

4.5.3 Teste de Simulação de Logs Sintéticos

Descrição: Teste da geração de casos sintéticos utilizando Discrete Event Simulation baseada na Rede de Petri descoberta.

Resultado Esperado: Geração de log sintético em formato XES com número de casos e eventos conforme parâmetros configurados, mantendo estrutura temporal similar ao log original.

Funcionalidade Validada: Etapa 3 do pipeline - Simulação de Logs Sintéticos, incluindo configuração da simulação, geração de casos e produção de logs de saída.

4.5.4 Teste de Validação de Qualidade

Descrição: Teste da comparação entre log original e sintético através de alinhamento de traces e cálculo de métricas de similaridade.

Resultado Esperado: Métricas de alinhamento indicando similaridade adequada (fitness médio > 0.7) entre logs original e sintético.

Funcionalidade Validada: Etapa 4 do pipeline - Validação de Qualidade, incluindo alinhamento de traces e cálculo de métricas de similaridade.

4.5.5 Teste de Integração Completa

Descrição: Teste de ponta-a-ponta executando o pipeline completo com log de exemplo (running-example.xes).

Resultado Esperado: Execução bem-sucedida de todas as etapas sem erros, produção de resultados dentro dos ranges esperados e tempo de execução aceitável.

Funcionalidade Validada: Integração completa entre todas as etapas do pipeline, validando fluxo de dados e arquitetura geral do sistema.

4.5.6 Teste de Interface de Usuário

Descrição: Teste da interface Streamlit incluindo upload de arquivos, navegação entre tabs, visualização de resultados e download de arquivos gerados.

Resultado Esperado: Interface responsiva e intuitiva, upload e processamento de arquivos XES, visualização adequada de métricas e diagramas, download funcional dos resultados.

Funcionalidade Validada: Interface de usuário completa, incluindo gerenciamento de estado, visualização de resultados e interação com o pipeline através da interface web.

4.6 Resultados Esperados e Obtidos

4.6.1 Resultados Esperados

O projeto estabeleceu metas para desenvolver um sistema automatizado de geração de modelos de simulação a partir de logs XES. Esperava-se um pipeline completo automatizado que executasse análise, mineração, simulação e validação de forma sequencial. O sistema deveria detectar automaticamente atributos essenciais como atividade, timestamp e case ID, produzir modelos de processo com fitness superior a 0.7 e gerar logs sintéticos com similaridade acima de 0.7 ao original.

Em termos de performance, esperava-se tempos de execução aceitáveis para uso interativo (inferiores a 5 minutos) e funcionamento em múltiplos domínios sem necessidade de adaptações específicas. A interface deveria ser intuitiva para usuários não especialistas, facilitando o acesso às técnicas de process mining.

4.6.2 Resultados Obtidos

O sistema implementado alcançou todos os objetivos principais estabelecidos. A automatização completa foi implementada com sucesso, resultando em um pipeline totalmente automatizado que processa logs de diferentes domínios sem modificações. Os logs sintéticos gerados apresentam similaridade superior a 80% aos originais, superando as expectativas iniciais.

A performance do sistema demonstrou-se adequada para uso interativo, com tempos de execução compatíveis com aplicações práticas. A interface web desenvolvida com Streamlit provou-se intuitiva e funcional, permitindo que usuários não especialistas utilizem o sistema efetivamente. A generalidade foi comprovada através do processamento bem-sucedido de logs de diferentes domínios organizacionais.

4.6.3 Limitações Identificadas

Durante o desenvolvimento, algumas limitações foram identificadas que não comprometem a funcionalidade principal do sistema. A escolha de transições na simulação permanece uniforme ao acaso, não incorporando probabilidades históricas de decisão. Distribuições multimodais de durações não são capturadas adequadamente pelos algoritmos de ajuste estatístico implementados.

O gerenciamento de recursos permanece simplificado, sem consideração de capacidade finita ou disponibilidade. A validação de logs grandes apresenta complexidade quadrática que limita a escalabilidade para datasets muito extensos. Essas limitações representam oportunidades claras para trabalhos futuros e não impedem o uso efetivo do sistema em cenários típicos.

5 Conclusão

Este trabalho apresentou o desenvolvimento de um gerador automatizado de modelos de simulação baseado em mineração de processos, demonstrando a viabilidade da integração entre essas duas tecnologias para apoiar a tomada de decisão operacional no curto prazo. O sistema desenvolvido representa uma contribuição significativa para o campo de process mining aplicado, oferecendo uma solução prática e generalizável para a criação automatizada de modelos de simulação.

5.1 Limitações e Desafios

Durante o desenvolvimento, foram identificadas limitações que representam oportunidades para trabalhos futuros. A simulação não incorpora probabilidades históricas de escolha de transições, utilizando seleção uniforme ao acaso. O ajuste estatístico de distribuições não captura adequadamente comportamentos multimodais, limitando a precisão temporal em alguns casos específicos.

O gerenciamento de recursos permanece simplificado, sem consideração de capacidade finita ou disponibilidade. A validação de logs extensos apresenta complexidade quadrática que pode impactar a performance em datasets muito grandes. Um desafio significativo foi lidar com a qualidade variável dos logs de entrada, que frequentemente apresentam dados incompletos, timestamps inconsistentes e nomenclaturas não padronizadas, exigindo mecanismos robustos de pré-processamento e filtragem. Essas limitações não comprometem a funcionalidade principal do sistema, mas indicam direções para aprimoramentos futuros.

5.2 Considerações Finais

O presente trabalho demonstrou que a integração automatizada entre mineração de processos e simulação de eventos discretos é não apenas tecnicamente viável, mas também prática e útil para análise e otimização de processos organizacionais.

Os resultados obtidos validam a viabilidade da abordagem e demonstram seu potencial para transformar a forma como organizações utilizam dados históricos para análise preditiva e otimização operacional. O sistema está pronto para aplicação em contextos educacionais e pode servir como base para pesquisas futuras em áreas relacionadas, contribuindo para o desenvolvimento contínuo de soluções inovadoras para desafios organizacionais complexos.

Referências

AALST, W. M. P. van der. *Process Mining: Data Science in Action*. 2nd. ed. Berlin: Springer, 2016. Citado 8 vezes nas páginas 16, 17, 19, 22, 23, 24, 37 e 39.

AALST, W. M. P. van der et al. Process mining manifesto. In: *Business Process Management Workshops (BPM 2011)*. [S.l.]: Springer, 2012. (Lecture Notes in Business Information Processing, v. 99), p. 169–194. Citado na página 23.

AUGUSTO, V. et al. Evaluation of discovered clinical pathways using process mining and joint agent-based discrete-event simulation. In: IEEE. *Proceedings of the Winter Simulation Conference (WSC)*. Washington, DC, USA, 2016. p. 2135–2146. Citado na página 25.

BRZYCHCZY, E.; ŻUBER, A.; AALST, W. van der. Process mining of mining processes: Analyzing longwall coal excavation using event data. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, v. 54, n. 5, p. 2723–2739, 2024. Citado na página 13.

FERRONATO, J. J. *PM4SOS: Um Framework para suporte à tomada de decisão operacional*. Tese (Doutorado em Informática) — Pontifícia Universidade Católica do Paraná (PUCPR), Curitiba, 2022. Disponível em: <https://www.ppgia.pucpr.br/pt/arquivos/doutorado/teses/2022/Tese_Jair_Jose_Ferronato.pdf>. Acesso em: 6 out. 2025. Citado 3 vezes nas páginas 12, 13 e 15.

FERRONATO, J. J.; SCALABRIN, E. E. Pm2sim: The automated creation of a simulation model from process mining. In: *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. Melbourne, Australia: IEEE, 2021. Citado 7 vezes nas páginas 17, 18, 20, 21, 22, 25 e 29.

GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, MA: Addison-Wesley Professional, 1994. ISBN 0-201-63361-2. Citado na página 63.

IEEE Computational Intelligence Society. *IEEE Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams*. 2010. IEEE Standard 1849-2016. Disponível em: <http://www.xes-standard.org>. Citado 3 vezes nas páginas 17, 29 e 34.

KHERBOUCHE, M. O.; LAGA, N.; MASSE, P.-A. Towards a better assessment of event logs quality. In: *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. Toronto, Canada: IEEE, 2020. p. 2235–2242. Citado 3 vezes nas páginas 17, 24 e 25.

LEEMANS, S. J. J.; FAHLAND, D.; AALST, W. M. P. van der. Discovering block-structured process models from event logs containing infrequent behaviour. In: *Proceedings of the 11th International Conference on Business Process Management*. [S.l.]: Springer, 2013. (Lecture Notes in Computer Science, v. 8094), p. 66–78. Citado 5 vezes nas páginas 17, 18, 19, 22 e 37.

LIU, S. T. *Integrating Process Mining with Discrete-Event Simulation Modeling*. Dissertação (Master of Science Thesis) — Brigham Young University, Provo, UT, 2015. Disponível em: <https://scholarsarchive.byu.edu/etd/5735>. Citado 3 vezes nas páginas 21, 22 e 24.

MARUŞTER, L.; BEEST, N. R. T. P. van. Redesigning business processes: A methodology based on simulation and process mining techniques. *Knowledge and Information Systems*, v. 21, n. 3, p. 267–297, 2009. Citado 3 vezes nas páginas 13, 15 e 16.

MENG, S. et al. Enhancing mine groundwater system prediction: Full-process simulation of mining-induced spatio-temporal variations in hydraulic conductivities via modularized modeling. *International Journal of Mining Science and Technology*, v. 34, p. 1625–1642, 2024. Citado na página 13.

NAKAJIMA, S. *Introduction to TPM: Total Productive Maintenance*. Cambridge, MA: Productivity Press, 1988. ISBN 0-915299-23-2. Citado na página 26.

PETERSON, J. L. *Petri Net Theory and the Modeling of Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1981. Citado 3 vezes nas páginas 18, 19 e 43.

SOUZA, T.; VACCARO, G. L. R.; LIMA, R. M. Operating room effectiveness: A lean health-care performance indicator. *International Journal of Lean Six Sigma*, ahead-of-print, 2020. Citado 3 vezes nas páginas 26, 28 e 51.

STROPARO, J. R.; BICHINHO, G. L.; PROTIL, R. M. Estudo da taxa de ocupação do centro cirúrgico através da modelagem e simulação de sistemas. *Anais do Congresso Brasileiro de Informática em Saúde*, PUCPR, Curitiba, 2004. Disponível em: <https://www.researchgate.net/publication/228848883>. Citado 2 vezes nas páginas 25 e 29.

WUENNENBERG, M.; WEGERICH, B.; FOTTNER, J. Towards data management and data science for internal logistics systems using process mining and discrete-event simulation. In: ELSEVIER B.V. *Procedia CIRP*. [S.l.], 2023. v. 120, p. 852–857. Citado 3 vezes nas páginas 13, 15 e 21.