UNIVERSIDADE ESTADUAL PAULISTA

"JÚLIO DE MESQUITA FILHO"

CAMPUS DE SÃO JOSÉ DO RIO PRETO

ANDRÉ ANTUNES MARQUES

Uma comparação entre bibliotecas de simulação de eventos discretos em diferentes linguagens de programação

São José do Rio Preto

2024

André Antunes Marques

Uma comparação entre bibliotecas de simulação de eventos discretos em diferentes linguagens de programação

Trabalho de Conclusão de Curso apresentado ao Conselho de Curso de Graduação em Ciência da Computação do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estatual Paulista, como parte dos requisitos para obtenção do diploma de Graduação em Ciência da Computação .

Orientadora:

Profa. Dra. Renata Spolon Lobato

São José do Rio Preto

2024

M357c

Marques, André Antunes
Uma comparação entre bibliotecas de simulação de eventos discretos em diferentes linguagens de programação / André Antunes Marques. -- São José do Rio Preto, 2024
43 p.

Trabalho de conclusão de curso (Bacharelado - Ciência da Computação) - Universidade Estadual Paulista (UNESP), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto
Orientadora: Renata Spolon Lobato

1. Simulação. 2. Simulação de Eventos Discretos. 3. Avaliação de Desempenho. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Dados fornecidos pelo autor(a).

André Antunes Marques

Uma comparação entre bibliotecas de simulação de eventos discretos em diferentes linguagens de programação

Trabalho de Conclusão de Curso apresentado ao Conselho de Curso de Graduação em Ciência da Computação do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estatual Paulista, como parte dos requisitos para obtenção do diploma de Graduação em Ciência da Computação .

Banca Avaliadora:

Profa. Dra. Renata Spolon Lobato
UNESP – Câmpus de São José do Rio Preto

Prof. Dr. Aleardo Manacero Junior
UNESP – Câmpus de São José do Rio Preto

Prof. Dr. Rodrigo Capobianco Guido
UNESP – Câmpus de São José do Rio Preto

São José do Rio Preto
22 de Outubro de 2024

Às minhas irmãs.

Agradecimentos

Agradeço ao meu pai, minha mãe e irmãs pelo apoio.

Agradeço à Profa. Dra. Renata Spolon Lobato pela atenciosidade em todos os passos.

Agradeço às minhas amigas pelo encorajamento.

Agradeço aos demais professores e colegas da UNESP de São José do Rio Preto.

Agradeço ao coral da UNESP do qual fui feliz de participar por toda a graduação.

“Ninguém nos pode impor, meu irmão, o que é melhor pra gente.“

(Milton Nascimento)

RESUMO

A Simulação de Eventos Discretos é uma técnica que permite avaliar sistemas que não poderiam ser testados no mundo real. Entre os meios de implementar um modelo de simulação está o uso de bibliotecas dedicadas à simulação. Existem várias bibliotecas de simulação que estão acessíveis para o público geral, desde aquelas voltadas à simulação de redes de computadores àquelas mais generalistas. Este trabalho faz uma comparação entre quatro bibliotecas de linguagens de programação distintas: SimPy, ConcurrentSim, DESMO-J e SMPL. O objetivo é trazer dados quantitativos à respeito do desempenho das bibliotecas e alguns fatores de qualidade de software. Para testar o desempenho são utilizados modelos da literatura que possuem implementação em SMPL e que, portanto, podem ser validados. Com o auxílio de *scripts* e ferramentas *open source* foi possível aferir medidas de desempenho e métricas estáticas dos códigos de cada modelo. No total foram utilizados dois modelos e os resultados obtidos apontaram quantitativamente uma eficiência de desempenho melhor para o SMPL enquanto que, para o ConcurrentSim e SimPy, os resultados apontaram uma melhor facilidade de entendimento do código. Os dados quantitativos podem ser úteis para estudantes, pesquisadores ou empresas que venham a utilizar uma das bibliotecas testadas.

PALAVRAS-CHAVE: Simulação, Avaliação de Desempenho

ABSTRACT

Discrete Event Simulation is a technique that allows the evaluation of systems that couldn't be tested in the real world. Among the ways of implementing a simulation model there are the simulation dedicated libraries. There are many simulation libraries in reach of the general public, some are more specific like the ones dedicated to simulate computer networks and some are more generalist. This work makes a comparison between 4 distinct open source libraries: SimPy, ConcurrentSim, DESMO-J and SMPL. The main goal is to get quantitative data regarding the performance of the libraries and some other factors of software quality. For the purpose of testing the performance of the libraries this work used models from the literature that had already been implemented in SMPL therefore could be validated. Performance measures and static metrics were taken with the assistance of scripts and open source tools. In total this work used 2 models for testing the libraries and the results pointed to a better performance for SMPL while ConcurrentSim and SimPy got better results for source code understanding. In general, the quantitative data can be useful for students, researchers and companies that may use one of the tested libraries.

KEYWORDS: Simulation, Performance Evaluation

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

Capítulo 1

Introdução

A simulação é uma técnica utilizada para imitar o funcionamento de diversos sistemas do mundo real (BANKS, 2014). Ela serve em casos que os sistemas reais não podem ser testados na prática e em que se deseja fazer um estudo que apresente a melhor maneira de implementar um projeto. Os sistemas simulados podem ser de diversas naturezas, desde simulações de guerra a simulações de sistemas computacionais.

Normalmente as simulações computacionais são realizadas através de uma das três ferramentas: simuladores, linguagens de simulação e bibliotecas e *frameworks* de linguagens de programação de uso geral (BANKS, 2014). Para este trabalho definiu-se que serão avaliadas as bibliotecas e *frameworks* de linguagens de uso geral voltadas à simulação de eventos discretos.

Dentre os simuladores disponíveis no mercado, destacam-se o Arena e o Anylogic, que possuem interfaces intuitivas e amigáveis, permitindo a modelagem de sistemas de forma visual e com pouca necessidade de habilidades em programação. No entanto, é importante ressaltar que esses simuladores são implementados em linguagens de programação, como o Java e o C++, assim como várias bibliotecas e *frameworks* para a simulação de eventos discretos.

No contexto acadêmico, o desenvolvimento de pesquisas e trabalhos científicos na área podem passar pela utilização de bibliotecas de simulação de eventos discretos. Nesse sentido é importante que as bibliotecas ofereçam uma série de funcionalidades para a implementação de modelos, o que facilita e agiliza o processo de implementação ou até melhora o desempenho dele.

Nesse sentido, o presente trabalho de conclusão de curso, ao comparar diferentes bibliotecas de simulação, promove a discussão e análise a respeito dessas bibliotecas, podendo ser útil para pesquisadores e desenvolvedores ao explorar algumas características das bibliotecas.

1.1 OBJETIVOS

É esperado que para simulações de sistemas com menos carga não haja diferença significativa considerando o desempenho dos *softwares* de simulação em diferentes linguagens. No entanto, em sistemas com mais carga pode ser que o tempo de execução e outros medidores de desempenho de *software* representem diferenças significativas que impactem no tempo que os desenvolvedores e analistas levam para iterar sobre os modelos experimentados. Dessa forma, o objetivo deste estudo é entender se há diferença significativa entre utilizar uma biblioteca ou outra do ponto de vista de desempenho e escalabilidade.

Tendo em vista que a diferença (ainda que pequena) de desempenho pode não ser somente resultado da linguagem utilizada, o outro objetivo é investigar se do ponto de vista da qualidade de *software* existe disparidade nas métricas de *software* obtidas buscando entender se há vantagem em termos de qualidade de *software* em se usar uma ou outra biblioteca. Os objetivos específicos são:

1. Comparar as bibliotecas considerando métricas de *software* (PRESSMAN; MAXIM, 2020).

2. Comparar o desempenho e escalbiliade das bibliotecas utilizando os modelos de Myron MacDougall (MACDOUGALL, 1987).

1.2 METODOLOGIA

A comparação das bibliotecas foi dividida em duas etapas:

1. Comparação considerando métricas estáticas (PRESSMAN; MAXIM, 2020).

2. Comparação de desempenho e escalabilidade (LIU, 2008).

Para a primeira etapa deste estudo, obteve-se as métricas através de ferramentas *open source* utilizadas sobre os códigos produzidos para os modelos. Para a segunda etapa deste estudo foram utilizados modelos cuja implementação em cada biblioteca pode ser verificada em (MACDOUGALL, 1987) e (ULSON, 2001) para aferir as medidas de desempenho.

Capítulo 2

Fundamentação Teórica

Este capítulo traz um preâmbulo do que é simulação, exemplos atuais de sua aplicabilidade bem como exemplos de trabalhos similares a este. Também são introduzidos algoritmos de geração de números pseudo-aleatórios e variáveis aleatórias.

No decorrer deste capítulo são introduzidas as técnicas de modelagem mais utilizadas com foco na modelagem de redes de filas que é a técnica na qual os modelos utilizados por este trabalho são descritos. São relembrados também alguns fatores de qualidade de *software* e apresentado o ângulo pelo qual este trabalho avalia a qualidade de *software* das bibliotecas testadas.

2.1 SIMULAÇÃO

A simulação é uma técnica para imitar o funcionamento de sistemas complexos que não poderiam ser testados no mundo real, segundo Banks (2014) a simulação envolve a geração de uma história artificial do sistema real para inferir seu comportamento ao longo do tempo. A partir desta história são inferidas características do sistema e relações entre suas entidades para se criar um modelo do sistema. Segundo Banks (2014) em alguns casos esses modelos podem ser resolvidos através de métodos simplesmente matemáticos, no entanto, muitos sistemas do mundo real são muito complexos e precisam, portanto, das simulações computacionais.

Em linhas gerais, a simulação é útil para entender o comportamento de um sistema ao longo do tempo oferecendo hipóteses de como as variáveis desse sistema do mundo real se comportam permitindo, segundo Banks (2014), o desenvolvimento de inovações desde novos *designs* de *hardware* a novos sistemas de transporte. Apesar disso, é preciso aferir a praticabilidade do uso de simulações visto que em alguns casos ela pode ser mais cara que a aplicação real ou

impossível de ser aplicada. Por exemplo, simular o comportamento humano em alguns casos não é uma tarefa considerada viável por ser muito complexo.

As simulações também podem ser utilizadas para a avaliação de desempenho de sistemas computacionais. A vantagem da simulação nesse caso é que ela pode ser utilizada em estágios iniciais do desenvolvimento enquanto as outras técnicas de aferição precisam do sistema em estágio final ou já implementado (SPOLON, 2010).

Na prática, a simulação computacional foi utilizada em diversos contextos ao longo do tempo e ainda segue como uma técnica promissora para diversos fins. Em Shoaib e Ramamohan (2022), por exemplo, são apresentados alguns modelos que representam operações no sistema de saúde primário da Índia. Já em Ibrahim et al. (2022) um modelo é utilizado para analisar o comportamento de multidões em contexto de evacuação.

Para abranger esta grande quantidade de contextos de uso, as simulações devem atender a dois tipos de sistemas: discretos e contínuos. Segundo Law (2014), mesmo nos casos em que os sitemas são discretos e contínuos ao mesmo tempo, na maioria dos casos, eles são predominantemente um dos tipos e portanto podem ser classificados no tipo predominante. O que define o tipo do sistema é o comportamento das variáveis de estado.

Para definir as variáveis de estado, no entanto, é necessário delimitar quais são os componentes desse sistema, ou seja, quais são as entidades presentes no sistema, quais atributos pertencem a essas entidades, quais são as atividades realizadas e quais eventos ocorrem. Tomando um exemplo, em Banks (2014) é sugerido um sistema de um banco no qual os clientes podem ser definidos como as entidades. Cada cliente também pode ter um saldo na conta corrente, este saldo é definido como um atributo da entidade cliente. Este cliente, ao chegar no banco, pode, por exemplo, realizar um depósito que seria uma atividade. Além disso, a própria chegada e partida do cliente podem ser consideradas eventos. Nesse sistema hipotético, portanto, as variáveis de estado poderiam ser a quantidade de caixas ocupados e a quantidade de clientes esperando atendimento.

A suposição desses componentes é uma peça chave na construção do modelo do sistema. Cabe ao construtor do modelo entender, no âmbito do que está sendo estudado, quais características são importantes e quais não são. A primeira linha da Tabela 1 traz os componentes propostos em Banks (2014) para o sistema hipotético do banco, a segunda linha são exemplos de componentes que poderiam também ser adicionados nesse modelo.

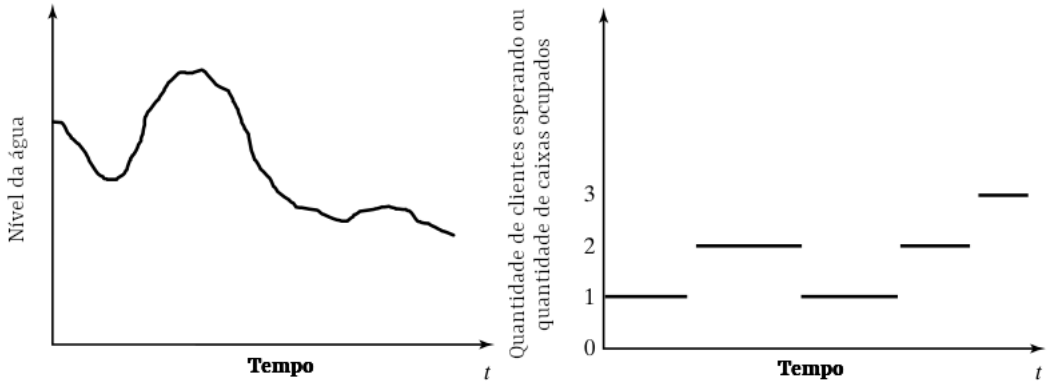
As variáveis de estado são dados quantitativos do sistema que mudam ao longo do tempo e

Tabela 1 – Componentes do Sistema Hipotético do Banco

Entidades	Atributos	Atividades	Eventos	Variáveis de Estado
Clientes	Saldo na Conta	Realizar Depósito	Chegada e partida	Clientes Esperando Caixas Ocupados
Funcionários	Cargo	Operar Caixa	Abertura e Fechamento	Funcionários Ociosos

fonte: Banks, adaptado e traduzido.

Figura 1 – Variáveis de Estado ao longo do Tempo nos Sistema Hipotéticos da Barragem e do Banco



fonte: Banks, adaptado e traduzido.

que possuem algum valor para o estudo realizado. No caso do sistema hipotético do banco as variáveis de estado denotadas na Tabela 1 são valores discretos, afinal, só se alteram no momento em que um cliente chega ou é atendido e não continuamente ao longo do tempo. Sendo assim o sistema é caracterizado como discreto.

Há também os sistemas nos quais as variáveis de estado se alteram de maneira contínua. Em Banks (2014) é dado o exemplo de uma barragem na qual uma variável de estado é o nível da água. Em caso de chuva, por exemplo, o nível vai variar de maneira contínua. Caso fossem tomadas medições ao longo do tempo, a imagem do lado esquerdo da Figura 1 representaria o nível da represa. Em comparação à direita está o gráfico para uma das variáveis de estado para o sistema do banco.

2.2 SIMULAÇÃO DE EVENTOS DISCRETOS

O escopo deste trabalho é a simulação de eventos discretos, ou seja, as simulações na qual o sistema estudado possui eventos que ocorrem de maneira discreta. De acordo com Banks (2014) alguns conceitos se repetem, ainda que com nomes diferentes, em todos os simuladores, linguagens de simulação, *frameworks* e bibliotecas de simulação, que nesta seção serão chamados de pacotes de simulação. Alguns desses conceitos já foram apresentados como as entidades, as

variáveis de estado, o sistema, o modelo do sistema, os atributos das entidades, os eventos e as atividades. Nesta seção serão retomados e explorados outros conceitos presentes em Banks (2014) bem como a relação entre eles. De acordo com Banks (2014) os conceitos que se repetem são: Sistema, Modelo, Variáveis de Estado, Entidade, Atributos, Lista, Evento, Registro de Evento, Lista de Eventos, Atividade, Atraso e Relógio.

Começando pelas atividades que costumam ter um tempo de duração que pode ser determinístico, estatístico ou dependente de alguma característica do sistema. Caso seja determinístico, ele é um tempo de duração fixo. Caso seja estatístico, ele varia em um conjunto de valores seguindo uma distribuição de probabilidade. Caso seja dependente de alguma característica do sistema, ele pode variar dependendo, por exemplo de um atributo específico de uma entidade que, quando alterado, altera o tempo de duração da atividade.

O tempo de duração das atividades, no entanto, é diferente do conceito de atraso. Tomando novamente o exemplo do banco e considerando os componentes da Tabela 1, o tempo de duração da atividade é o tempo que um cliente leva para realizar um depósito. Nesse mesmo exemplo, o atraso poderia ser o tempo que um cliente espera na fila para realizar um depósito. Ou seja, o atraso é uma espera condicionada, nesse caso, à quantidade de clientes na frente da fila de clientes e ao tempo que cada cliente demora para realizar o depósito.

Segundo Banks (2014), um evento é gerenciado colocando-se um registro do evento na lista de eventos futuros. O registro de evento contém no mínimo o tipo de evento e o tempo em que este evento ocorrerá. Usando o exemplo da Tabela 1 o tipo de evento pode ser "Chegada", "Partida", "Abertura"ou "Fechamento". A lista de eventos futuros contém os registros de evento ordenados em ordem de ocorrência na simulação.

Outro conceito considerado recorrente por Banks (2014) são as listas. Elas podem ser definidas como uma coleção de entidades associadas ordenadas seguindo alguma lógica. Valendo-se do exemplo da Tabela 1, uma lista pode ser a fila de clientes esperando para realizar um depósito. Nesse exemplo, ela está ordenada por ordem de chegada, ou seja, segue a disciplina FIFO (*first-in, first-out*).

Para finalizar o vocabulário de conceitos presentes em Banks (2014) tem-se o conceito de relógio que nada mais é que a variável que representa o tempo simulado. Com esses conceitos presentes em todos os simuladores e linguagens de simulação torna-se possível o escalonamento de eventos, que é o método para simular eventos que permite avançar no relógio de evento em evento baseado na lista de eventos futuros.

No sistema do mundo real certamente não é possível avançar no tempo, no entanto, como o tempo de cada evento simulado já está registrado na lista de eventos futuros, isso torna-se possíivel no escalonamento de eventos. Nas seções 2.3 e 2.4 serão apresentados os métodos de geração de números aleatórios e distribuições de probabilidade que permitem gerar os valores de tempo para os registros de evento de forma a simular a aleatoriedade do mundo real.

No decorrer da execução da simulação os eventos são adicionados e retirados da lista de eventos futuros, tornando necessário o gerenciamento dessas operações que é chamado de processamento de lista. Um pacote de simulação que realiza o processamento de lista de forma eficiente gera um impacto direto no tempo de processamento do computador (BANKS, 2014).

Em linguagens procedurais de uso geral o processamento dessas listas é feito utilizando tanto listas dinâmicas quanto estáticas, que são atualizadas durante o tempo de execução, ou seja, os registros de eventos são adicionados e retirados da lista de eventos futuros em tempo de execução. Nos casos em que os registros devem ser gerenciados no final ou começo da lista, o gasto de processamento não é grande. No entanto, quando as operações devem ser feitas no meio da lista, a necessidade de processamento é aumentada devido a necessidade de busca na lista. Para lidar com esse problema, uma alternativa consiste em criar ponteiros no meio da lista nos quais o algoritmo pode iniciar a busca, diminuindo assim o tamanho da lista percorrida. Ou então utilizar outras estruturas de dados como, por exemplo, árvores e *heaps* (BANKS, 2014).

2.3 GERAÇÃO DE NÚMEROS ALEATÓRIOS

A geração de números aleatórios é um recurso fundamental na simulação de eventos discretos. Com ela é possível simular sistemas estocásticos, ou seja, que possuem comportamento variável no mundo real. Caso um analista quisesse simular um hospital, por exemplo, os pacientes desse hospital chegam em tempos não relacionados, ou seja, o tempo entre as chegadas dos pacientes é aleatório. Nesse e em outros casos são necessários métodos para a geração computacional de números pseudo-aleatórios (BANKS, 2014).

Métodos chamados de pseudo-aleatórios não são considerados totalmente aleatórios pois, quando reutilizados com os mesmos parâmetros, geram o mesmo conjunto de números ainda que esses números não tenham relação entre si. O objetivo geral é gerar números independentes, ou seja, que não possuem relação e gerar números entre 0 e 1 sendo que cada número tem a mesma probabilidade de ser gerado, em outras palavras, segue uma distribuição de probabilidade uniforme para a geração desses números (BANKS, 2014).

Um exemplo de geração de números pseudo-aleatórios é o método de congruência linear. Nessa técnica são utilizados os parâmetros X_0 como a semente, a como multiplicador, c como incremento e m como o divisor da operação módulo (mod). A relação que gera a sequência de números inteiros $X_1, X_2, X_3...$ é a seguinte:

$$X_{i+1} = (a * X_i + c) \mod m, i = 0, 1, 2, ...$$

Estes números inteiros são então convertidos para valores entre 0 e 1 para gerar os números psudo-aleatórios R_i seguindo a relação:

$$R_i = \frac{X_i}{m}, i = 1, 2, 3, ...$$

Em Banks (2014) é utilizado um exemplo com os parâmetros $X_0 = 27, a = 17, c = 43, m = 100$ para os quais os valores de X_i e R_i obtidos são:

$$X_0 = 27$$

$$X_1 = (17 * 27 + 43) \mod 100 = 502 \mod 100 = 2$$

$$R_1 = \frac{2}{100} = 0.02$$

$$X_2 = (17 * 2 + 43) \mod 100 = 77 \mod 100 = 77$$

$$R_2 = \frac{77}{100} = 0.77$$

$$...$$

Quando um número da sequência X_i se repete a geração de números aleatórios entrará em um ciclo repetindo todos os números novamente. Na prática, os valores dos parâmetros podem ser otimizados para obter a maior sequência sem repetição e com a menor distância entre cada número gerado. Segundo Banks (2014) escolher um valor de m que seja uma potência de 2 beneficia a velocidade e a eficiência da geração de números aleatórios em um computador.

Outras formas de gerar números pseudo-aleatórios foram desenvolvidas e com o aumento do poder computacional puderam ser melhoradas. A técnica Mersenne Twister é um exemplo de técnica que representa melhora em relação à congruência linear na geração de números pseudo-aleatórios (MATSUMOTO; NISHIMURA, 1998).

Para determinar a qualidade dos geradores de números pseudo-aleatórios foram criados os

testes de de frequência e autocorrelação. Esses testes servem para experimentar os principais critérios da geração de números pseudo-aleatórios que são a distribuição uniforme dos números gerados e a independência.

2.4 GERAÇÃO DE VARIÁVEIS ALEATÓRIAS

Na seção 2.3 foi feito um preâmbulo geral da geração de números pseudo-aleatórios trazendo a visão de Banks (2014) sobre a utilização da geração de números pseudo-aleatórios. Segundo ele, os sistemas do mundo real apresentam variáveis aleatórias que podem ser simuladas através da geração de números pseudo-aleatórios. O tempo entre chegadas é uma dessas variáveis de um sistema que pode ser considerada aleatória. No entanto, na maioria dos sistemas simulados, essas variáveis de caráter aleatório seguem um padrão de distribuição probabilístico não necessariamente uniforme.

Em alguns casos os dados de entrada para as variáveis já existem de observações do mundo real de outros estudos, nesses casos é melhor utilizar estes dados existentes. No entanto, quando esses dados são escassos ou não existem cabe ao analista ou equipe de analistas saber avaliar qual distribuição de probabilidade é mais adequada para cada variável aleatória(LAW, 2014).

Segundo Law (2014) há uma variedade de distribuições de probabilidade que podem ser úteis para as simulações. Como as variáveis aleatórias podem ser tanto contínuas quanto discretas, as distribuições de probabilidade também podem ser contínuas ou discretas, afinal, para cada valor possível da variável existe uma probabilidade dele acontecer.

No caso das variáveis aleatórias discretas a função que relaciona um possível valor da variável aleatória com a probabilidade desse valor ocorrer se chama função massa de probabilidade (FMP). Para as variáveis contínuas a função que denota a mesma relação se chama função densidade de probabilidade (FDP).

A partir dessas funções os pacotes de simulação conseguem transformar uma sequência R_i que segue uma distribuição uniforme, em uma sequência que segue uma distribuição de probabilidade específica. Um dos algoritmos que realizam esse trabalho é o da transformação inversa. Esse algoritmo utiliza a função inversa da FDP e da FMP para gerar a sequência seguindo algumas distribuição específica (BANKS, 2014).

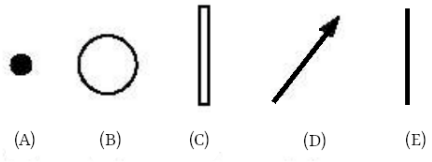
2.5 TÉCNICAS DE MODELAGEM

Esta seção será dedicada a explicar algumas técnicas de modelagem utilizadas na modelagem de sistemas computacionais. Algumas das técnicas elencadas em Spolon (2010) utilizadas para simulação de sistemas computacionais são: Redes de Petri e Redes de Fila.

As Redes de Petri, segundo Spolon (2010), são representadas por dois componentes básicos: um ativo chamado transição e um passivo chamado lugar. A transição é representada por uma barra e diz respeito a ações realizadas pelo sistema. O lugar é representado por um círculo e diz respeito às variáveis de estado. Esses dois componentes são ligados por arcos. Há também os *tokens* que, segundo Spolon (2010), representam objetos que transitam pela rede e aguardam pelo processamento nos lugares.

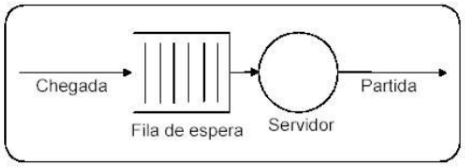
Ao longo do tempo surgiu a demanda de se associar variáveis aleatórias às transições na rede de Petri tradicional. Foi desenvolvida então a Rede de Petri Estocástica Generalizada, que possui dois tipos de transação: temporizada e não temporizada. A temporizada é aquela associada a uma variável aleatória e dispara após um tempo aleatório exponencialmente distribuído (representada por um retângulo) e a não temporizada que dispara em tempo zero quando está habilitada (representada por um traço) (Figura 2).

Figura 2 – Rede de Petri: (a) Token, (b) Lugar, (c) Transição Temporizada, (d) Arco, (e) Transição não Temporizada



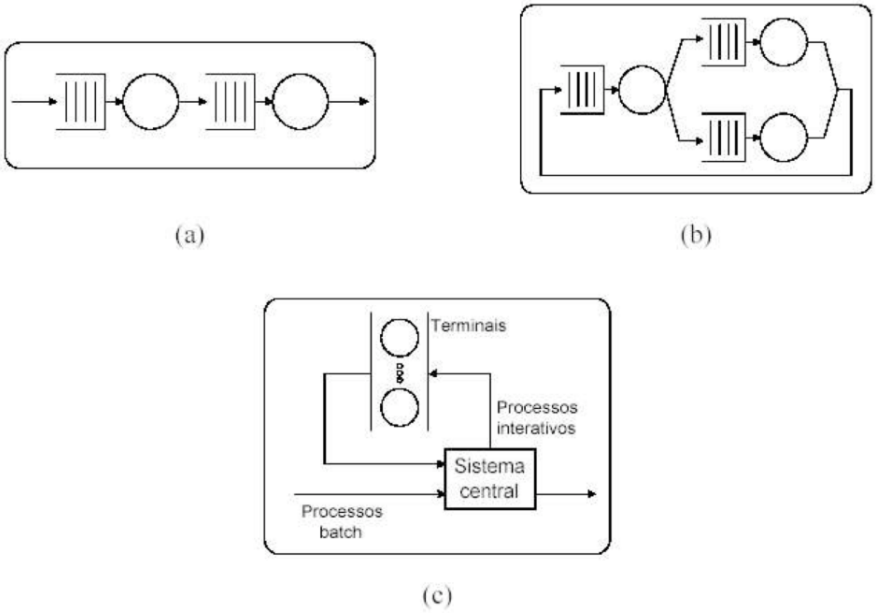
fonte: adaptado de Spolon (2010).

Figura 3 – Centro de Serviço



fonte: retirado de Spolon (2010).

Figura 4 – Tipos de Redes (a) Aberta, (b) Fechada, (c) Mista



fonte: retirado de Spolon (2010).

Para finalizar as técincas de modelagem serão introduzidas as Redes de Filas. A formação de filas ocorre quando há disputa de usuários para utilizar o mesmo recurso dentro de um sistema. A fila é então formada pelos usuários que esperam um tempo até que chegue sua vez. Dessa forma, os componentes presentes em todas as redes de filas são: os servidores que são entidades que atendem uma fila, as filas que são listas que organizam os clientes e os clientes que são entidades que demandam algum serviço.

De acordo com Spolon (2010), o conjunto de um ou mais servidores e uma ou mais filas é chamado centro de serviço e o conjunto de centros de serviço forma uma rede de filas. A Figura 3 traz a representação visual dos componentes do centro de serviço.

Conforme ilustrado na Figura 4 retirada de Spolon (2010) as redes de fila podem ser classificadas como abertas, fechadas ou mistas. Em redes abertas, os clientes chegam e saem do sistema de forma dinâmica. Já em redes fechadas, o número de clientes é constante e eles circulam pelo sistema, sendo que nenhum cliente entra ou sai. Por fim, redes mistas combinam elementos das redes abertas e fechadas, ou seja, possuem tanto clientes que circulam pelo sistema quanto chegadas e saídas de clientes.

A disciplina da fila refere-se à política ou regra que determina como os clientes são atendidos no sistema. A disciplina mais comum é a FIFO, onde os clientes são atendidos na ordem em que chegaram. No entanto, em situações onde os clientes têm diferentes prioridades ou características, podem ser utilizadas políticas de priorização para determinar quem será atendido primeiro. A

notação padrão para um sistema de filas é $A/S/c/K/m/SD$, onde A representa a distribuição do tempo de chegada, S representa a distribuição do tempo de serviço, c é o número de servidores, K é o número máximo de clientes no sistema, m é o número de clientes disponíveis na fonte e SD é a disciplina da fila.

Neste trabalho escolheu-se utilizar modelos de redes de filas que simulam sistemas computacionais, para comparar as bibliotecas.

2.6 QUALIDADE DE SOFTWARE

A qualidade de *software* está atrelada a dois fundamentos: satisfação do usuário final e conformidade com os requisitos do produto. Ao longo dos anos, alguns fatores de qualidade de *software* foram propostos. No sentido de satisfazer o usuário final os fatores, segundo Pressman e Maxim (2020) são: eficácia, eficiência, satisfação, ausência de riscos, cobertura de contexto.

No que atende aos requisitos do produto deve-se levar em conta: adequação funcional, eficiência de desempenho, compatibilidade, usabilidade, confiabilidade, segurança, manutenibilidade, portabilidade (PRESSMAN; MAXIM, 2020).

Para avaliar boa parte destes fatores, principalmente no que tange a satisfação do usuário, seria necessário realizar testes com usuários finais, o que não será feito neste trabalho. Além disso, testes e métricas devem ser aferidos ao longo de todo ciclo de vida de um produto de software e, portanto, este trabalho não apresenta solução absoluta para os fatores avaliados. Dessa forma, o que se faz neste trabalho é, a partir de testes e métricas, avaliar fatores de qualidade de software que poderiam ser melhorados.

Segundo Pressman e Maxim (2020) a avaliação de qualidade de *software* pode ser tanto qualitativa quanto quantitativa. A usabilidade, por exemplo, é um fator que tende a ser mais qualitativo que quantitativo. Por outro lado, uma abordagem quantitativa pode ser útil para indicar muitos problemas de *software* através de métricas. As métricas podem ser úteis para identificar alto acoplamento ou complexidade desnecessária no código, por exemplo. No entanto, vale ressaltar que não há uma métrica que indique a qualidade de *software* de maneira total e exata mas sim características que estão presentes em *softwares* de qualidade.

De maneira geral, para garantir a qualidade de *software* os desenvolvedores podem se apoiar em práticas como testes, revisões de código ou ferramentas de análise estática. Segundo Pressman e Maxim (2020) um código "bom o suficiente"varia de acordo com o interesse da organização que o está desenvolvendo e portanto no caso de organizações que produzem um *software* com

finalidade comercial vale aferir até que ponto essas práticas de garantia de qualidade devem ser adotadas.

No contexto dos *softwares open source* não há necessariamente a lógica de custo-benefício aplicada no desenvlvimento de um *software*. Sendo assim, a qualidade do *software* está mais atrelada às necessidades da comunidade que o mantém e desenvolve. No artigo (KHATAMI; ZAIDMAN, 2023) é feito um estudo com 471 contribuidores de grandes projetos *open source* na plataforma GitHub. O estudo contou com questionários para avaliar o conhecimento dos contribuidores a respeito de práticas em qualidade de *software*. A conclusão foi de que havia um nível alto de conhecimento por parte dos participantes a respeito das práticas mas também uma incerteza em como elas deveriam ser aplicadas.

Dentre os fatores que indicam qualidade de um *software* este trabalho irá tratar principalmente da eficiência de desempenho. No entanto, sob a luz de algumas métricas estáticas, pontua algumas questões de manutenibilidade. Essas métricas podem ser obtidas dos códigos fonte dos modelos escolhidos para este estudo.

As métricas estáticas oferecem informações a respeito do *software* que podem indicar uma redundância dentro do código que o deixa mais difícil de entender. Algumas métricas estáticas podem ser extraídas dos códigos fonte dos modelos em cada biblioteca para indicar a qualidade do código. Algumas métricas utilizadas neste estudo são:

1. **Linhas de Código (LOC):** É uma medida do tamanho do código. Arquivos grandes podem ser mais difíceis de gerenciar.

2. **Contagem de Funções/Métodos:** Refere-se ao número total de blocos de código que são definidos como funções ou métodos.

3. **Contagem de Tokens:** Refere-se ao número total de unidades léxicas individuais no código-fonte. Uma alta contagem de tokens pode indicar maior dificuldade de entendimento do código.

4. **Complexidade Ciclomática:** É uma métrica quantitativa da complexidade do código que mede o número de caminhos independentes por meio do código-fonte. Geralmente, funções com uma complexidade ciclomática alta são mais propensas a erros.

Apesar de poderem ter uma relação indireta com eficiência de desempenho do *software*, essas métricas estáticas não são um indicativo de desempenho. A eficiência de desempenho de

software refere-se à eficiência com que um *software* executa suas funções, medido em termos de velocidade, capacidade de resposta, estabilidade e uso de recursos. A partir do desempenho é possível aferir a escalabilidade que se refere ao desempenho sob o aumento da carga no *software*.

Segundo Sommerville (2011) testes de desempenho estão interessados tanto em demonstrar que o sistema atende seus requisitos quanto em descobrir problemas e defeitos do sistema. A melhor forma de fazer isso seria através de testes de estresse no qual os limites do *software* são explorados. De maneira geral nos testes de estresse o *software* recebe uma sobrecarga para que o avaliador possa entender seu comportamento quando sobrecarregado.

Segundo Liu (2008) os três fatores que mais impactam o desempenho e a escalabilidade de um sistema de *software* são: a capacidade do *hardware* no qual o *software* está sendo rodado, a maturidade da plataforma de *software* subjacente (como sistema operacional, *drivers* de interface de dispositivo, ambiente de execução) e o próprio *design* e implementação do *software*. Por isso, em todo teste de desempenho devem ser levados em conta a máquina (os componentes como processador, memória primária e secundária, etc) e o sistema operacional.

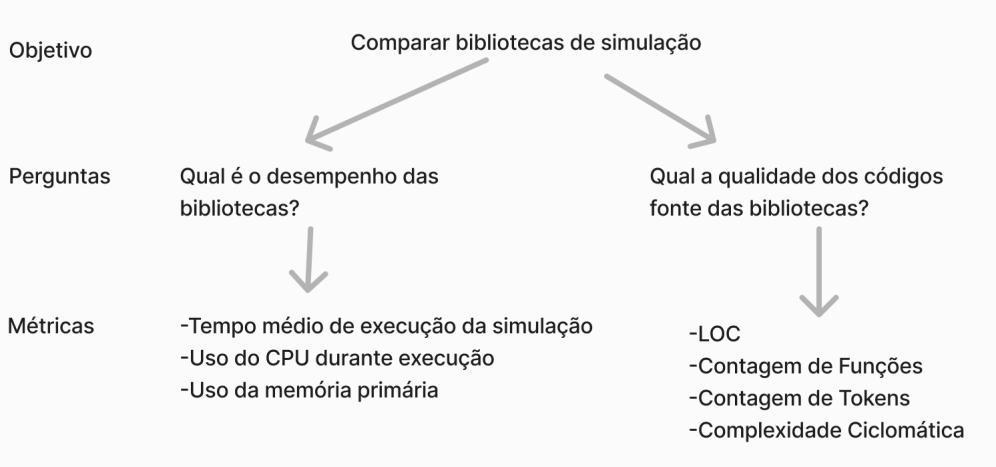
De acordo com Liu (2008) para medir desempenho normalmente são considerados quatro fatores principais: o quão ocupada a unidade central de processamento (CPU) fica executando o *software*, o quanto da memória primária é utilizada durante o teste, o quão ocupada fica a memória secundária e o quão ocupada a rede fica (quando há utilização das redes). Estes fatores podem ser aferidos através de *system performance counters* que são programas que estão presentes em todos os sistemas operacionais populares.

2.7 PARADIGMA *GOAL-QUESTION-METRIC*

Segundo Fenton e Bieman (2014) a medição é útil apenas se ajudar a entender um aspecto relevante do *software*. Dessa forma, determinar os atributos apropriados para medir depende dos objetivos da avaliação do *software*.

Sendo assim, o primeiro passo é determinar os objetivos do estudo. Em seguida, devem ser geradas as perguntas para verificar se os objetivos foram cumpridos. Finalmente, cada pergunta deve ser analisada para identificar as medições necessárias para responder a cada uma delas. O paradigma para definição de métricas que segue estes passos é o *Goal-Question-Metric* (GQM). Na Figura 5 é apresentado o diagrama GQM que definem as métricas definidas para este trabalho.

Figura 5 – Diagrama *Goal-Question-Metric* do objetivo deste estudo



fonte: Elaborado pelo autor.

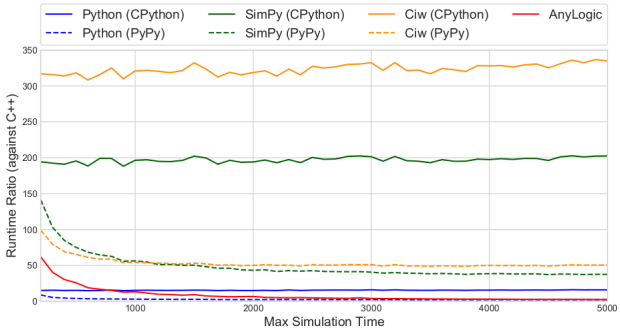
2.8 TRABALHOS CORRELATOS

1. **"Ciw: An open source discrete event simulation library":** Este trabalho apresenta o Ciw como uma alternativa de biblioteca *open source* em Python para conduzir simulações de eventos discretos. Ao longo do trabalho, é feita uma comparação da biblioteca Ciw com outras bibliotecas e *frameworks* populares no ponto de vista de desempenho e adequação para simulação de eventos discretos. Em termos de tempo de execução foram comparados cinco *softwares*: Python, C++, SimPy, Ciw e Anylogic com um mesmo modelo variando o tempo de simulação. Como esperado C++, por ser uma linguagem compilada, foi o mais rápido. Segundo os autores era esperado que utilizar a linguagem padrão em vez de uma biblioteca também diminuísse o tempo de execução.

   Na Figura 6, retirada de (PALMER et al., 2019), vê-se que o tempo de execução da Ciw utilizando o interpretador CPython foi entre 300 e 350 vezes o do C++, enquanto SimPy estava em torno de 200 vezes. Fica visível também na Figura 6 que a razão do tempo de execução utilizando o interpretador PyPy é mais afetada para valores menores de tempo máximo de simulação provavelmente devido ao fato de que o tempo que o código leva para ser interpretado se torna menos relevante em relação ao tempo total da execução conforme o tempo de execução aumenta devido ao aumento de carga (aumento do tempo de simulação).

   Para os autores, no entanto, a prioridade não era o tempo de execução e sim outros fatores de qualidade de *software* como a facilidade de leitura do código fonte.

Figura 6 – A imagem é um gráfico retirado de (PALMER et al., 2019) no qual é feita uma comparação do tempo de execução de um mesmo modelo variando o tempo máximo de simulação de 200 a 5000 em diferentes *softwares*. Cada execução foi realizada cinco vezes e o tempo de execução médio foi dividido pelo tempo de execução em C++. Em pontilhado foi utilizando o interpretador PyPy, em linha contínua utilizando o interpretador CPython.

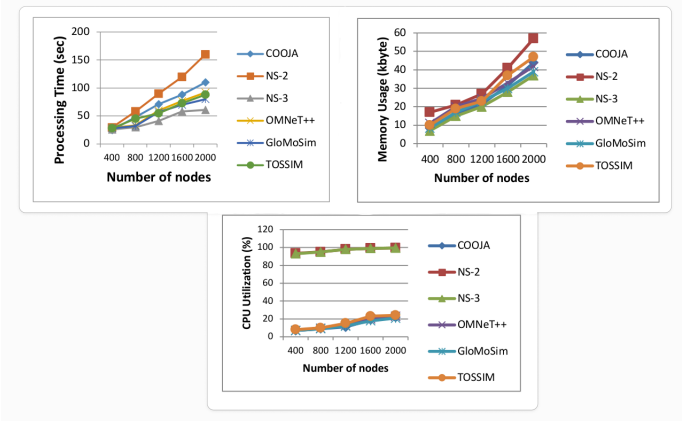


fonte: retirado de (PALMER et al., 2019)

2. **"On the Performance and Scalability of Simulators for Improving Security and Safety of Smart Cities":** O escopo deste artigo é a simulação voltada às cidades inteligentes. O estudo é em formato de *survey* e compara os simuladores: COOJA, NS-2 com o *framework* Mannasim, NS-3, OMNeT++ com o *framework* Castalia, WSNet, TOSSIM, J-Sim, GloMoSim, SENSE, e Avrora. A maioria são simuladores de eventos discretos voltados à simulação de redes. A maior pate deles é implementada em C,C++ ou Java (MOHSIN et al., 2022).

   Ao longo do artigo são feitas comparações considerando o desempenho do *software* de simulação e quais componentes e protocolos de diferentes camadas possuem suporte para modelar no *software*. Para a comparação de desempenho foi utilizado um modelo de rede variando de 400 a 2000 nós. Para o mesmo modelo em cada *software* foram medidos, entre outras métricas, o tempo de execução em segundos, a porcentagem de uso da CPU e a quantidade de memória utilizada em kilobytes. Na Figura 7, retirada de (MOHSIN et al., 2022), estão as médias dos experimentos realizados 8 vezes para cada *software* em cada uma das métricas mencionadas.

Figura 7 – Três gráficos contendo as métricas de desempenho de cada *software* testado: tempo de execução em segundos, a porcentagem de uso da CPU e a quantidade de memória utilizada em kilobytes. O número de nós vai de 400 a 2000.

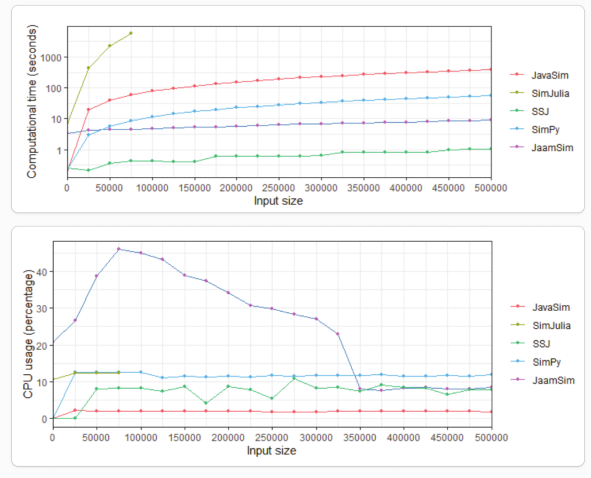


fonte: retirado de (MOHSIN et al., 2022)

3. **"Experimental Comparison of Open Source Discrete-Event Simulation Frameworks":** Este artigo traz um compilado de *frameworks* de simulação de eventos discretos que tiveram atualização recente à publicação do estudo. Em termos de usabilidade é feita uma avaliação informal e em termos de desempenho é feita uma avaliação considerando as métricas de tempo de execução e porcentagem de uso da CPU.

   Neste estudo é utilizada uma rede de petri. Para cada *framework* foi testada a carga no intervalo de 25000 a 500000 na rede de petri proposta (Figura 8). Em linhas gerais o estudo traz um comparativo em termos quantitativos e qualitativos entre os *frameworks* (KRISTIANSEN et al., 2022).

Figura 8 – Os gráficos representam o tempo computacional em segundos e a percentagem de uso da CPU em relação ao aumento de carga em cada *framework*



fonte: retirado de (KRISTIANSEN et al., 2022)

Capítulo 3

Desenvolvimento

Este capítulo apresenta as ferramentas utilizadas e os métodos para aferir as métricas definidas no diagrama *"Goal-Question-Metric"* da Figura 5. Também são apresentados os modelos utilizados e o método de validação bem como as métricas do relatório da biblioteca SMPL. Por fim são definidas as classes que precisaram ser desenvolvidas para cada implementação.

O critério escolhido para definir as bibliotecas avaliadas foi: uma biblioteca *open source* de quatro linguagens de uso geral distintas (Python, Java, C e Julia), no caso, SimPy[1], DESMO-J[2], SMPL (MACDOUGALL, 1987) e ConcurrentSim[3]. Já o critério de definição das métricas a serem aferidas seguiu o paradigma *Goal-Question-Metric* definido na seção 2.7.

3.1 TESTE DE DESEMPENHO E ESCALABILIDADE

O teste de desempenho leva em conta três métricas: porcentagem de uso da CPU, tempo de execução e tamanho da memória principal utilizada durante a execução. As métricas foram aferidas para cada um dos dois modelos de fila descritos na seção 3.1.1 utilizando ferramentas de medição nativas do linux[4]. O objetivo da implementação foi obter um relatório final igual ao da extensão funcional SMPL para que a implementação pudesse ser validada. O relatório da extensão funcional SMPL é descrito na seção 3.1.2.

O critério de escolha dos modelos utilizados no teste de desempenho foi: dois modelos que possuem resultado no livro (MACDOUGALL, 1987) ou em (SPOLON, 1999) para a validação, ou seja, o Sistema de Servidor Único e o Sistema Computacional Simples. Para testar o comportamento das bibliotecas foi feita uma progressão de carga que segue um aumento

[1] https://gitlab.com/team-simpy/simpy/
[2] https://desmoj.sourceforge.net/home.html
[3] https://github.com/JuliaDynamics/ConcurrentSim.jl
[4] https://www.man7.org/linux/man-pages/man1/time.1.html

exponencial para cada modelo. No caso do Sistema de Servidor Único a carga testada é o tempo de simulação e no caso do Sistema Computacional Simples é a quantidade de tarefas que deixam o sistema. A Tabela 2 contém as informações do computador no qual os softwares foram testados.

Tabela 2 – Características do Computador Utilizado

Sistema Operacional	Ubuntu 22.04.4 LTS
Processador	Intel® Core™ i7-8700 3.20GHz × 6
RAM	16,0 GiB
SSD	240,1 GB

fonte: Elaborado pelo autor.

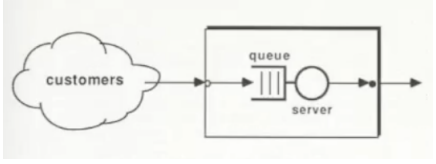
3.1.1 Modelos Utilizados no Trabalho

Os modelos utilizados simulam sistemas computacionais através de redes de filas. É comum estudos utilizarem um ou mais modelos para testar o comportamento de um *software* relacionado a simulação de eventos discretos como em (SPOLON, 1999). Os valores de parâmetros que serão citados nas próximas seções, como tempo de simulação, são apenas os valores iniciais para cada modelo. São utilizados diversos valores de parâmetros para cada modelo e é mostrada a imagem final do sistema para cada valor, o que permite a validação de cada implementação em cada biblioteca utilizada neste trabalho (MACDOUGALL, 1987).

- **Sistema de Servidor Único:**

Neste sistema os clientes chegam em um tempo que segue uma distribuição exponencial com uma média de 200 unidades de tempo, e o servidor executa o serviço também em um tempo que segue uma distribuição exponencial mas com ma média de 100 unidades de tempo. O tempo da simulação é de 200000 unidades de tempo. O sistema é aberto e segue uma disciplina na fila no qual o primeiro cliente a chegar é atendido primeiro. Sendo assim a notação padrão para esse sistema é $M/M/1/\infty/\infty$/FIFO, onde M indica uma distribuição exponencial. O diagrama que representa este modelo é a Figura 9.

Figura 9 – Diagrama do Sistema de Servidor Único

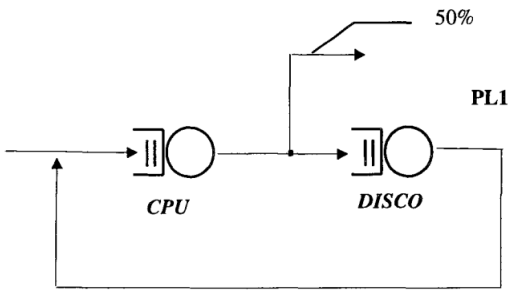


fonte: retirado de (MACDOUGALL, 1987)

- **Sistema Computacional Simples:**

Neste sistema há uma CPU e um disco. Em Spolon (1999) é utilizada uma taxa de chegada das tarefas na CPU que segue uma distribuição exponencial com média 50 unidades de tempo. O tempo de serviço na CPU segue uma distribuição exponencial com uma média de 10 unidades de tempo. Após o tempo de serviço na CPU a tarefa pode deixar o sistema ou solicitar o serviço do disco. O tempo de serviço no disco segue uma distribuição exponencial com média 20 unidades de tempo. Após sair do disco a tarefa retorna à fila da CPU. A probabilidade da tarefa sair do sistema é de 50%, logo, a probabilidade de ir para o disco também é 50%. A validação desse modelo nas bibliotecas utilizadas no presente trabalho foi feita a partir da implementação presente em (ULSON, 2001). Originalmente o modelo foi implementado utilizando o SMPL que é a uma extensão funcional que implementa simulações em C.

Figura 10 – Diagrama do Sistema Computacional



fonte: retirado de (SPOLON, 1999)

3.1.2 Validação

O SMPL é uma extensão funcional da linguagem C voltada à simulação de eventos discretos apresentada em (MACDOUGALL, 1987). Todos os modelos escolhidos para este trabalho possuem implementações em SMPL disponíveis em (MACDOUGALL, 1987) ou (ULSON, 2001). Dessa forma, a validação dos modelos implementados nas bibliotecas avaliadas por este trabalho se dá pela comparação do relatório gerado pelo SMPL para cada simulação com os dados gerados em cada implementação de cada biblioteca.

No relatório gerado pelo SMPL os dados gerados são relacionados aos centros de serviço no modelo. Para exemplificar, um centro de serviço pode ser um recurso do sistema modelado. Sendo assim, os dados gerados no relatório são definidos como:

- ***Utilization:*** É obtida pela divisão do tempo de simulação em que o centro de serviço ficou ocupado pelo tempo total coberto pelo relatório.

- ***Mean Busy Time:*** É a divisão do tempo em que o centro de serviço ficou ocupado pela quantidade de vezes em que o centro de serviço foi liberado seja pelo término do serviço ou por interrupção durante o tempo coberto pelo relatório.
- ***Mean Queue Length:*** É o somatório do tempo que as tarefas ficaram na fila com o tempo de serviço. Esse valor é dividido pelo tempo coberto pelo relatório. Em seguida é subtraído o valor de *Utilization*.
- ***Releases:*** É a quantidade de vezes que um centro de serviço foi liberado seja por preempção ou término do serviço durante o período coberto pelo relatório.
- ***Preemptions:*** É a quantidade de vezes que o centro de serviço sofreu uma preempção ao longo do tempo coberto pelo relatório.
- ***Queue:*** É a quantidade de requisições que foram enfileiradas e atendidas pelo centro de serviço incluindo a de tarefas que foram atendidas e retornaram à fila devido a uma preempção.

Na Tabela 3 estão os resultados do relatório do modelo do Servidor Único para cada biblioteca. Os resultados do relatório para o Sistema Computacional Simples estão nas Tabelas 4 e 5.

Tabela 3 – Valores gerados para o modelo de Servidor Único com Tempo de Simulação 200000.

	SMPL	Simpy	ConcurrentSim	DESMO-J
Utilization	0.5301	0.5301	0.5301	0.5301
Mean Busy Time	103.482	103.482	103.482	103.482
Mean Queue Length	0.659	0.659	0.659	0.659
Releases	1025	1025	1025	1025
Preemptions	0	0	0	0
Queue	558	558	558	558

fonte: Elaborado pelo autor.

Tabela 4 – Valores gerados para o CPU do modelo de Sistema Computacional Simples com 1000 tarefas deixando o sistema.

	SMPL	Simpy	ConcurrentSim	DESMO-J
Utilization	0.3903	0.3903	0.3903	0.3903
Mean Busy Time	9.735	9.735	9.735	9.735
Mean Queue Length	0.255	0.255	0.255	0.255
Releases	1982	1982	1982	1982
Preemptions	0	0	0	0
Queue	775	775	775	775

fonte: Elaborado pelo autor.

Tabela 5 – Valores gerados para o DISCO do modelo de Sistema Computacional Simples com 1000 tarefas deixando o sistema.

	SMPL	Simpy	ConcurrentSim	DESMO-J
Utilization	0.4001	0.4001	0.4001	0.4001
Mean Busy Time	20.188	20.188	20.188	20.188
Mean Queue Length	0.295	0.295	0.295	0.295
Releases	980	980	980	980
Preemptions	0	0	0	0
Queue	394	394	394	394

fonte: Elaborado pelo autor.

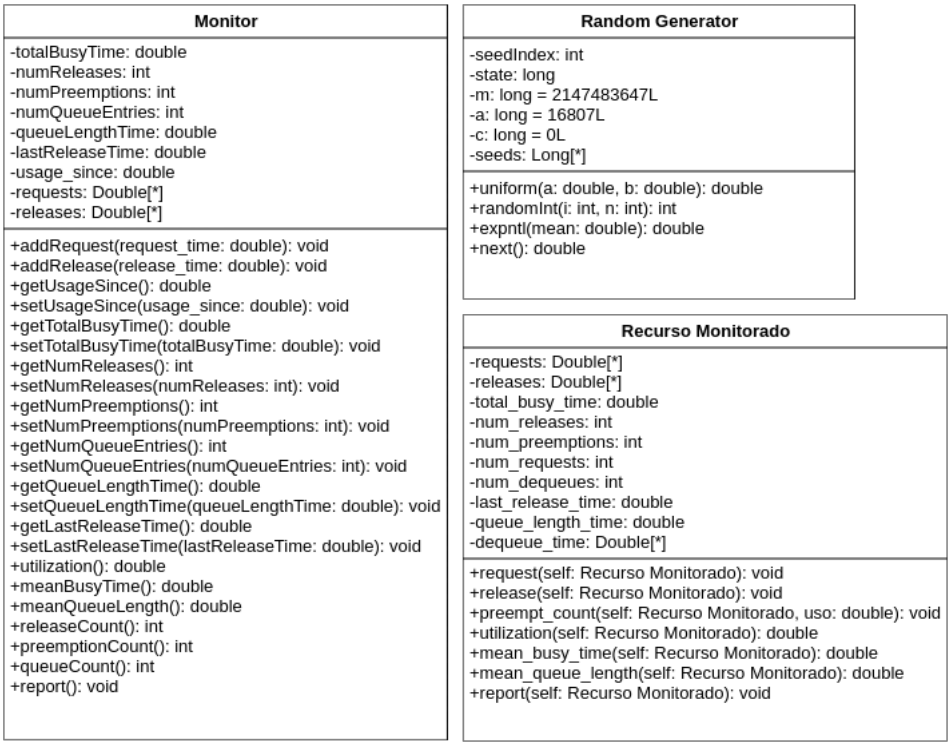
3.2 IMPLEMENTAÇÃO

Para ser possível validar as implementações dos modelos nas bibliotecas foi necessário desenvolver uma classe que gerasse um relatório com as mesmas métricas do SMPL e uma classe que gerasse números pseudo-aleatórios utilizando o mesmo algoritmo de geração de números pseudo-aleatórios do SMPL. No SimPy e no ConcurrentSim essa classe foi denominada "Recurso Monitorado". Ainda que o DESMO-J tivesse um relatório próprio foi necessário implementar uma classe denominada "Monitor"pois as métricas geradas no relatório original eram diferentes das métricas do SMPL.

Para que os valores obtidos fossem iguais em todos os relatórios foi necessário utilizar o mesmo gerador de números pseudo-aleatórios. Como as bibliotecas atuais utilizam o *Mersenne Twister* e o SMPL utiliza a congruência linear optou-se em implementar o algoritmo de congruência linear para todas as bibliotecas em uma classe denominada "Random Generator". Também foram implementadas as funções que geram os valores pseudo-aleatórios seguindo as distribuições: uniforme e exponencial.

Por fim, os modelos foram implementados tomando como base exemplos do tutorial e o manual de cada biblioteca. O diagrama apresentado na Figura 11 apresenta os tipos de dados dos atributos das classes criadas e o tipo de dado retornado pelos métodos das classes criadas.

Figura 11 – Diagrama que representa as classes criadas



fonte: Elaborado pelo autor.

Capítulo 4

Resultados

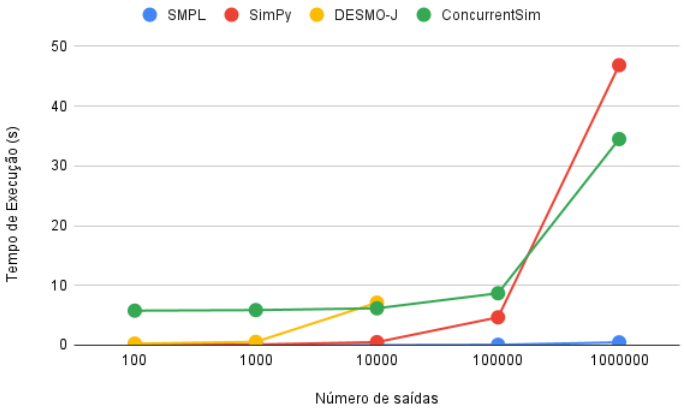
Neste capítulo são apresentados os resultados das métricas determinadas no diagrama daFigura 5. Além disso é feita uma comparação considerando as características gerais das bibliotecas que foram observadas durante o trabalho e que representam vantagens e desvantagens relevantes para o programador que as utiliza.

4.1 MÉTRICAS DE DESEMPENHO

Um dos fatores de qualidade de *software* que este trabalho propôs-se a avaliar é a eficiência de desempenho e escalabiblidade. Para aferir a eficiência de desempenho foram utilizadas as métricas determinadas no diagrama da Figura 5: tempo médio de execução (segundos), uso da CPU (%) e uso máximo da memória primária (kilobytes).

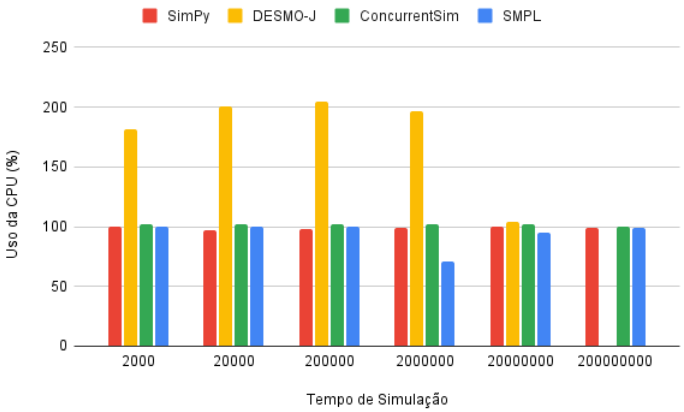
Para os valores de tempo de execução os resultados são a média de 5 experimentos com cada carga. Para o uso de CPU e pico de memória utilizada os resultados são a mediana de 5 experimentos com cada carga. Na Figura 12 são mostrados os resultados de tempo de execução dado o aumento do número de tarefas que deixam o Sistema Computacional Simples. Na Figura 13 são mostrados os resultados de uso da CPU dado o aumento do tempo de simulação para o Sistema de Servidor Único. Na Figura 14 são mostrados os resultados do pico de memória utilizada dado o aumento do tempo de simulação para o Sistema de Servidor Único.

Figura 12 – Tempo de execução (segundos) de cada software para o Sistema Computacional Simples variando a quantidade de tarefas que deixam o sistema (Número de Saídas).



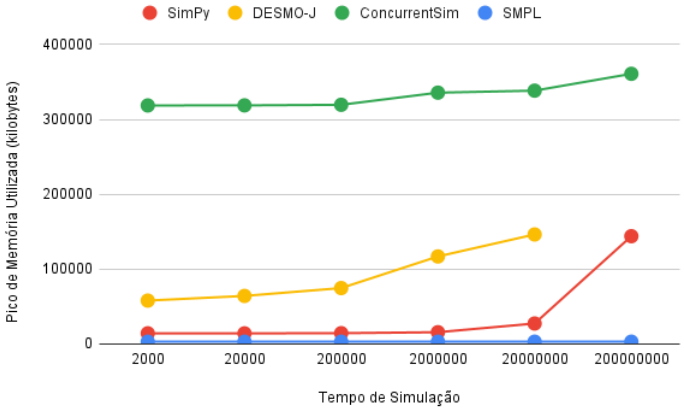
fonte: Elaborado pelo autor.

Figura 13 – Uso de CPU (%) para o Sistema de Servidor Único.



fonte: Elaborado pelo autor.

Figura 14 – Pico de memória utilizada (kilobytes) para o Sistema de Servidor Único.



fonte: Elaborado pelo autor.

Pelo gráfico da Figura 12 é possível aferir que, a partir de um Número de Saídaas próximo a 100000, o tempo de execução do código em SimPy utilizando o interpretador CPython tende a ser maior que o tempo de execução no ConcurrentSim. Também é possível observar que para a biblioteca DESMO-J não foi obtido um resultado para as duas maiores cargas.

Pelo gráfico da Figura 13 é possível aferir que, excetuando-se para a biblioteca DESMO-J, não houve tanta diferença no uso da CPU. A porcentagem chega a ser maior que 100% pois a CPU utilizada possui 6 núcleos, o que na teoria poderia gerar um uso do até 600% mas esse valor nunca é atingido devido aos outros processos que são executados ao mesmo tempo.

Pelo gráfico da Figura 14 percebe-se que ConcurrentSim é a biblioteca que possui os maiores picos de memória utilizada. Já a biblioteca SMPL mantém a mesma quantidade de memória em todas as execuções porque nela não é utilizada a alocação dinâmica de memória, ou seja, quando uma carga extrapola o tamanho máximo da lista de eventos futuros é necessário aumentá-la manualmente.

Em linhas gerais já era esperado que o SMPL consumisse menos recursos que as demais bibliotecas, mas para além disso é interessante observar quantitativamente o comportamento das outras bibliotecas de mais alto nível.

Para a biblioteca DESMO-J não foi possível aferir todas as medidas devido a um erro apresentado com o uso de cargas maiores. Para verificar que o erro não era proveniente da possível má implementação dos modelos foi utilizado um exemplo do tutorial do DESMO-J em um teste de estresse. Nesse teste de estresse foi aumentado o tempo de simulação e o mesmo erro foi apresentado. O computador utilizado no teste foi o mesmo da Tabela 2.

4.2 MÉTRICAS ESTÁTICAS

Uma maneira de se obter indicadores de qualidade de *software* é através de métricas estáticas. Essas métricas podem ser associadas a fatores como a manutenibilidade do código, afinal um código mais fácil de ler é mais fácil de entender e atualizar.

As métricas estáticas foram obtidas de maneira automatizada através da ferramenta *open source* Lizard[1] e através de *scripts*. A ferramenta permite obter as métricas: linhas de código, média de linhas de código por função, complexidade ciclomática média por função, média de tokens por função e contagem de funções. As métricas podem ser geradas tanto por arquivo

[1] https://github.com/terryyin/lizard

quanto por diretório. As tabelas 6 e 7 apresentam os resultados das métricas estáticas aferidos para cada modelo em cada biblioteca.

Observa-se pelos valores apresentados nas tabelas 6 e 7uma maior similaridade nas métricas obtidas dos modelos em ConcurrentSim e Simpy e uma maior discrepância para o DESMO-J e o SMPL. A vantagem de se utilizar uma biblioteca mais inteligível está na manutenção e entendimento do código fonte, nesse quesito as linguagens de mais alto nível se saem melhor. Considerando as métricas por função, por exemplo, fica visível a desvantagem do SMPL e considerando a quantidade de funções e linhas de código fica visível a desvantagem do DESMO-J.

Tabela 6 – Métricas estáticas para o Sistema de Servidor Único

Biblioteca	LOC	LOC/func	CC/func	Tokens/func	Qtd. Funções
SimPy	118	6.1	1.5	45.4	17
SMPL	28	27.0	6.0	171.0	1
DESMO-J	284	5.3	1.3	36.2	43
ConcurrentSim	107	5.2	1.3	50.8	13

fonte: Elaborado pelo autor.

Tabela 7 – Métricas estáticas para o Sistema Computacional Simples

Biblioteca	LOC	LOC/func	CC/func	Tokens/func	Qtd. Funções
SimPy	139	6.6	1.6	48.8	18
SMPL	46	45.0	10.0	284.0	1
DESMO-J	362	6.1	1.4	43.8	48
ConcurrentSim	154	6.3	1.5	55.4	16

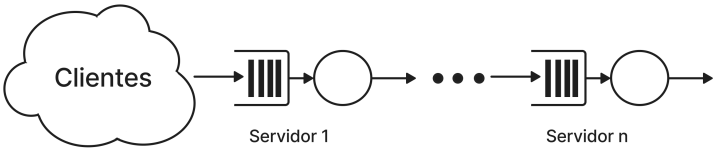
fonte: Elaborado pelo autor.

4.3 COMPARANDO SIMPY E CONCURRENTSIM

Dadas as similaridades nas métricas de desempenho e qualidade de *software* das bibliotecas SimPy e ConcurrentSim este trabalho propõe uma comparação mais detalhada entre as bibliotecas utilizando um terceiro modelo. Este modelo é denominado Servidores em Série.

No modelo dos Servidores em Série os clientes chegam em intervalos de tempo que seguem uma distribuição exponencial com uma média de 200 unidades de tempo. Neste modelo uma determinada quantidade de servidores estão postos em sequência e os clientes visitam um após o outro conforme ilustrado na Figura 15. Os servidores executam o serviço em um tempo que segue uma distribuição exponencial com média de 100 unidades de tempo. O tempo da simulação é de 2000000 de unidades de tempo.

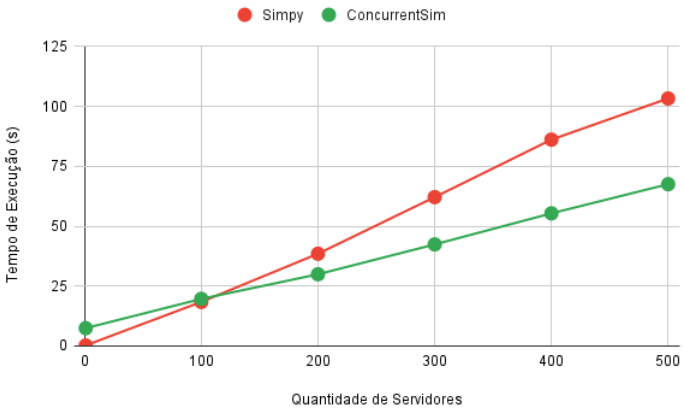
Figura 15 – Diagrama do Modelo de Servidores em Série



fonte: Elaborado pelo autor.

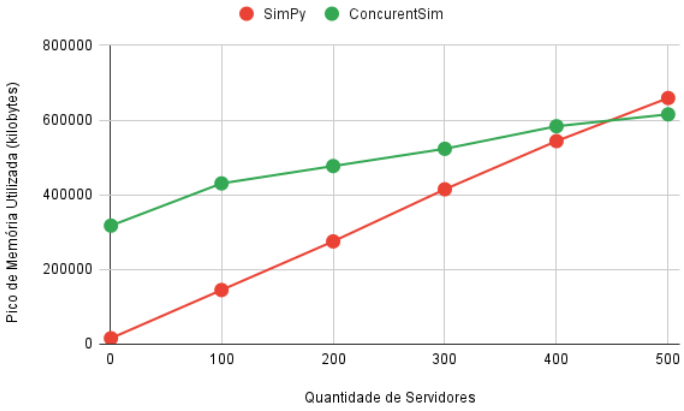
Para o experimento a carga utilizada foi a quantidade de servidores variando de 1 a 500. Na Figura 16 e na Figura 17 são apresentadas respectivamente a média e a mediana de 5 execuções considerando o tempo de execução e o pico de memória utilizada para cada biblioteca.

Figura 16 – Tempo de execução (segundos) para o Modelo de Servidores em Série.



fonte: Elaborado pelo autor.

Figura 17 – Pico de memória utilizada (kilobytes) para o Modelo dos Servidores em Série.



fonte: Elaborado pelo autor.

Neste experimento fica mais fácil de observar, em termos de escalabilidade, a vantagem do ConcurrentSim em relação ao SimPy utilizando o interpretador CPython considerando o tempo de execução. Já havia uma indicação pela Figura 14 que o aumento do pico de memória era mais acentuado para o Simpy que para o ConcurrentSim e pela Figura 17 fica mais fácil observar que a quantidade de memória utilizada pelo SimPy torna-se maior que pelo ConcurrentSim considerando o modelo dos Servidores em Série a partir de 500 servidores.

4.4 COMPARAÇÃO

Ao longo da implementação dos modelos, foram identificadas algumas características das bibliotecas que este trabalho considera relevante. Nesta seção são apresentadas essas características e discutidas suas vantagens e desvantagens para o programador que utiliza alguma das bibliotecas utilizadas neste trabalho. Na Tabela 8 são apresentadas quais características foram consideradas e qual a situação das bibliotecas avaliadas.

A primeira característica avaliada é o relatório gerado pela biblioteca. Ter um relatório padrão gerado dá a vantagem de economia de esforço do programador e uniformidade de resultados em diferentes simulações. Por outro lado é importante que haja certa flexibilidade para que, caso os dados apontados no relatório não sejam úteis no experimento, seja possível aferir outros tipos de resultados.

No caso deste trabalho, por exemplo, os dados gerados pelo relatório da biblioteca DESMO-J não foram suficientes, o que não impede que em futuros trabalhos sejam dados suficientes. Já as bibliotecas SimPy e ConcurrentSim não possuem relatório algum. Em todos os casos foi necessário desenvolver uma forma de captar e apresentar os dados gerados na simulação.

No que diz respeito à geração dos números pseudo-aleatórios as bibliotecas mais atuais utilizam o mesmo algoritmo. No entanto, a biblioteca SMPL é antecessora à criação da técnica do *Mersenne Twister* e utiliza a técnica de congruência linear. O *Mersenne Twister* no entanto, produz sequências de números pseudo-aleatórios maiores e mais uniformemente distribuídas. O Simpy, o ConcurrentSim e o DESMO-J utilizam o *Mersenne Twister*.

Considerando a alocação de memória, o uso de memória estática dá uma certa segurança ou previsibilidade de quanto de memória será utilizado na simulação, porém, caso o tamanho da lista de eventos futuros exceda o limite, a simulação não ocorrerá da maneira correta. Já o uso de alocação dinâmica torna a implementação mais prática, contudo, mais imprevisível. As bibliotecas mais atuais utilizam a alocação dinâmica.

Das bibliotecas avaliadas apenas a DESMO-J possui alguma interface gráfica. No caso, ela permite acompanhar a execução da simulação apenas. Outras bibliotecas de simulação não avaliadas neste trabalho permitem, além de visualizar a execução da simulação, implementar todo o modelo através de uma interface gráfica, por exemplo, a Jaamsim[2].

Para finalizar avaliou-se a dificuldade de programar em cada biblioteca. Para exemplificar, este trabalho considera a implementação de recursos em um modelo. Nas bibliotecas SimPy e ConcurrentSim essa implementação é muito similar: elas permitem a utilização de classes de recursos pré-codificados mas também é possível customizá-los. Já a biblioteca DESMO-J permite diferentes paradigmas ao implementar um modelo: por processos ou por eventos. No entanto, para todo recurso, seja ele encarado como um processo no paradigma de processos ou como uma entidade no paradigma de eventos, deve ser criada uma classe própria, o que aumenta o esforço de programação.

Tabela 8 – Características Gerais das Bibliotecas

Biblioteca	Relatório	Geração de NPA	Alocação de Memória	Interface Gráfica	Programação
SimPy	não possui	Mersenne Twister	Dinâmica	não possui	simples
SMPL	possui	Congruência Linear	Estática	não possui	simples
DESMO-J	possui	Mersenne Twister	Dinâmica	possui alguns	laboriosa
ConcurrentSim	não possui	Mersenne Twister	Dinâmica	não possui	simples

fonte: Elaborado pelo autor.

[2] https://jaamsim.com/

Capítulo 5

Conclusão e Trabalhos Futuros

Este trabalho conclui portanto que é necessário considerar o escopo de uso da biblioteca utilizada. Se for um propósito educativo há de se considerar algo mais fácil de entender como o SimPy ou o ConcurrentSim.

É importante ressaltar também a importância dos *softwares open source* cuja existência permite a realização de tantos trabalhos acadêmicos. Dessa forma, os dados quantitativos deste trabalho podem ser base para que trabalhos futuros tentem melhorar os resultados dessas bibliotecas.

Entre as dificuldades encontradas na realização deste trabalho é importante citar a tentativa de implementação do modelo do Servidor Central (MACDOUGALL, 1987). Este modelo lida com prioridade e preempção nos servidores. Este trabalho chegou a validar a implementação dele em SimPy mas não conseguiu validar uma implementação na biblioteca DESMO-J pois, apesar de oferecer suporte à interrupção, a biblioteca não possui um recurso preemptivo, ou seja, a preempção deve ser implementada manualmente. Por isso, uma das dificuldades encontradas foi o tempo para implementar e validar os modelos considerando as dificuldades em implementá-los.

Em trabalhos futuros modelos com preempção podem ser testados para explorar o comportamento das bibliotecas em diferentes casos. Além disso, para aferir mais fatores de qualidade podem ser realizados testes com usuários finais das bibliotecas.

REFERÊNCIAS

BANKS, J. **Discrete-event System Simulation**. [S.l.]: Pearson, 2014. (Always learning). ISBN 9781292024370.

FENTON, N.; BIEMAN, J. **Software Metrics: A Rigorous and Practical Approach, Third Edition**. [S.l.]: CRC Press, 2014. (Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series). ISBN 9781439838235.

IBRAHIM, A. M. et al. The role of crowd behavior and cooperation strategies during evacuation. **Simulation**, Society for Computer Simulation International, San Diego, CA, USA, v. 98, n. 9, p. 737–751, sep 2022. ISSN 0037-5497. Disponível em: https://doi.org/10.1177/00375497221075611.

KHATAMI, A.; ZAIDMAN, A. Quality assurance awareness in open source software projects on github. In: **2023 IEEE 23rd International Working Conference on Source Code Analysis and Manipulation (SCAM)**. Los Alamitos, CA, USA: IEEE Computer Society, 2023. p. 174–185. Disponível em: https://doi.ieeecomputersociety.org/10.1109/SCAM59687.2023.00027.

KRISTIANSEN, O. S. et al. Experimental comparison of open source discrete-event simulation frameworks. In: SPRINGER. **International Conference on Simulation Tools and Techniques**. [S.l.], 2022. p. 315–330.

LAW, A. **Simulation Modeling and Analysis**. [S.l.]: McGraw-Hill Education, 2014. ISBN 9780073401324.

LIU, H. **Software Performance and Scalability: A Quantitative Approach**. [S.l.]: Wiley, 2008. (Quantitative Software Engineering Series). ISBN 9780470465394.

MACDOUGALL, M. **Simulating Computer Systems: Techniques and Tools**. [S.l.]: MIT Press, 1987. (Computer Systems Series). ISBN 9780262132299.

MATSUMOTO, M.; NISHIMURA, T. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. **ACM Trans. Model. Comput. Simul.**, Association for Computing Machinery, New York, NY, USA, v. 8, n. 1, p. 3–30, jan. 1998. ISSN 1049-3301. Disponível em: https://doi.org/10.1145/272991.272995.

MOHSIN, A. et al. On the performance and scalability of simulators for improving security and safety of smart cities. In: **2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)**. [S.l.: s.n.], 2022. p. 1–8.

PALMER, G. I. et al. Ciw: An open-source discrete event simulation library. **Journal of Simulation**, Taylor & Francis, v. 13, n. 1, p. 68–82, 2019.

PRESSMAN, R.; MAXIM, B. **Software Engineering: A Practitioner's Approach**. [S.l.]: McGraw-Hill Education, 2020. ISBN 9781260548006.

SHOAIB, M.; RAMAMOHAN, V. Simulation modeling and analysis of primary health center operations. **Simulation**, Society for Computer Simulation International, San Diego, CA, USA, v. 98, n. 3, p. 183–208, mar 2022. ISSN 0037-5497. Disponível em: https://doi.org/10.1177/00375497211030931.

SOMMERVILLE, I. **Engenharia de software**. [S.l.]: Pearson Prentice Hall, 2011. ISBN 9788579361081.

SPOLON, R. **Simulação distribuída em plataformas de portabilidade: viabilidade de uso e comportamento do protocolo CMB**. Tese (Doutorado) — Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 1999. Disponível em: www.teses.usp.br.

SPOLON, R. **Simulação como Ferramenta para Avaliação de Sistemas Computacionais**. [S.l.]: Escola Regional de Informática, Rio de Janeiro, 2010.

ULSON, R. S. **PARSMPL e SMPLx Manual Técnico - Descrição e Estrutura de Dados**. [S.l.], 2001. Disponível em: https://www.icmc.usp.br/institucional/estrutura-administrativa/biblioteca/publicacoes/relatorios-tecnicos.