Universidade de Brasília
Faculdade de Tecnologia

Simulador de replay de mercado de bolsa de valores para trading de alta frequência

Matheus Santos Alves

PROJETO FINAL DE CURSO

ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Brasília

2023

Universidade de Brasília
Faculdade de Tecnologia

Simulador de replay de mercado de bolsa de valores para trading de alta frequência

Matheus Santos Alves

Projeto Final de Curso submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação

Orientador: Prof. Dr. Geovany Araújo Borges

Brasília
2023

Santos Alves, Matheus.
S237s Simulador de replay de mercado de bolsa de valores para trading de alta frequência / Matheus Santos Alves; orientador Geovany Araújo Borges. -- Brasília, 2023.
64 p.

Projeto Final de Curso (Engenharia de Controle e Automação) -- Universidade de Brasília, 2023.

1. Bolsas de valores. 2. Simulador de mercado. 3. Trading algorítmico. 4. High frequency trading. I. Borges, Geovany Araújo, orient. II. Título

Universidade de Brasília
Faculdade de Tecnologia

Simulador de replay de mercado de bolsa de valores para trading de alta frequência

Matheus Santos Alves

Projeto Final de Curso submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação

Trabalho aprovado. Brasília, 20 de dezembro de 2023:

Prof. Dr. Geovany Araújo Borges,
UnB/FT/ENE
Orientador

Profa. Dra. Genaina Nunes Rodrigues,
UnB/CIC
Examinadora interna

Prof. Dr. Edson Mintsu Hung,
UnB/FT/ENE
Examinador interno

Brasília
2023

Resumo

Em bolsas de valores eletrônicas, como a bolsa de valores brasileira (B3) é comum o emprego de estratégias de finanças baseadas em modelos matemáticos e estatísticos que visam otimizar o processo de tomada de decisão relacionado à compra e venda de ativos financeiros, com o objetivo final de maximizar os lucros obtidos. O emprego de estratégias não otimizadas diretamente em mercados reais pode ser extremamente prejudicial, já que o mau funcionamento de um modelo pode levar à grandes perdas monetárias. Como forma de minimizar este problema, utiliza-se de simuladores de mercado para testar e analisar estratégias de investimento *ex-post* antes de aplicá-las em mercados reais. No mercado de software atual é possível encontrar diversas plataformas de simulação financeira que funcionam como ferramentas educacionais, fornecendo funcionalidades básicas relacionadas à negociação de ativos, mas que não permitem a implementação de conceitos mais avançados de investimento, como o de *trading* algorítmico. Este trabalho contempla a concepção e o desenvolvimento de um software de simulação financeira capaz de reproduzir dinâmicas de mercado reais através de dados históricos e testar a performance de estratégias de investimento a partir da análise de lucros e perdas executadas sobre o mercado simulado. O objetivo do trabalho é fornecer uma ferramenta de software robusta e de propósito educacional, que permita aproximar pessoas com *background* técnico em programação à área de investimentos financeiros, sobretudo através do estudo de estratégias de alta frequência de negociação, como o *high frequency trading* (HFT). Ao fim do documento serão apresentados os resultados obtidos quando comparamos dados provenientes da execução de simulações com dados do mundo real. Por meio de análise gráfica iremos constatar que o simulador é capaz de aproximar de maneira relativamente satisfatória a dinâmica de mercado regida pelo livro de ofertas e pelas negociações ocorridas no mundo real.

Palavras-chave: Bolsas de valores. Simulador de mercado. Trading algorítmico. High frequency trading.

Abstract

In electronic stock exchanges, such as the Brazilian stock exchange (B3), it is common to employ finance strategies based on mathematical and statistical models aimed at optimizing the decision-making process related to the buying and selling of financial assets, ultimately aiming to maximize profits. Employing non-optimized strategies directly in real markets can be extremely detrimental, as a malfunctioning model can lead to significant monetary losses. To mitigate this issue, market simulators are used to test and analyze investment strategies *ex-post* before applying them in real markets. In today's software market, various financial simulation platforms exist as educational tools, offering basic functionalities related to asset trading but not allowing for the implementation of more advanced investment concepts, such as algorithmic trading. This work encompasses the design and development of financial simulation software capable of replicating real market dynamics using historical data and testing the performance of investment strategies through the analysis of gains and losses realized in the simulated market. The objective is to provide a robust educational software tool that bridges individuals with a technical programming background to the field of financial investments, particularly through the study of high-frequency trading strategies like high frequency trading (HFT). At the end of the document, the results obtained from comparing data derived from simulations with real-world data will be presented. Through graphical analysis, we will demonstrate that the simulator can relatively satisfactorily approximate the market dynamics governed by the order book and actual trades that took place in the real world.

Keywords: Stock exchange. Market simulator. Algorithmic Trading. High frequency trading.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Nos últimos anos, o número de investidores pessoa física da bolsa de valores brasileira (B3) aumentou significativamente. Em 2018, o número total de CPFs cadastrados na B3 era aproximadamente 700 mil, e no fim do ano de 2022 esse número ultrapassou a marca de 5 milhões, com um crescimento de aproximadamente 700% [1]. De acordo com uma pesquisa encomenda pela própria B3[2], uma das principais causas desse aumento se dá pela redução significativa na taxa de juros ao longo dos últimos anos, o que despertou um maior interesse em investimentos de renda variável, já que os lucros sobre investimentos de renda fixa foram drasticamente reduzidos. Aliado a este fator, a popularização das mídias digitais relacionadas a finanças pessoais, sobretudo em plataformas online, possuiu papel importantíssimo neste crescimento, e segundo esta mesma pesquisa, 73% dos participantes afirmaram buscar informações relacionadas a investimentos através de canais do *Youtube* e influenciadores digitais.

Em ambientes como o da B3, onde milhões de negociações são realizadas por dia, há uma complexa dinâmica de mercado, onde o preço de um ativo financeiro pode sofrer influência de diversos fatores externos, tornando praticamente impossível prever com precisão absoluta os movimentos de mercado futuros. Por conta dessa natureza não determinística, ao longo dos anos foram desenvolvidas diversas técnicas de investimento visando maximizar os lucros, e nas últimas décadas, com o auxílio de computadores, tornou-se possível avaliar estas estratégias computacionalmente, a partir de dados históricos de mercados reais.

Um dos campos mais relevantes que busca estudar e entender estes mercados é o de finanças quantitativas, que busca aplicar métodos matemáticos e estatísticos em mercados financeiros e no processo de investimento em sua totalidade. A partir da análise de dados e da modelagem de estratégias, busca-se otimizar o processo de tomada de decisão ao realizar novos investimentos.

Neste contexto, é natural pensar em ferramentas educacionais que permitem pôr em prática conceitos teóricos e estratégias de investimento em ambientes de simulação, onde as perdas e os riscos financeiros inerentes ao processo de investimento não são reais. O mercado de *softwares* com esta finalidade é bastante aquecido, e oferece ferramentas para todos os níveis de investidores, desde aqueles que estão começando a se aventurar no mundo de renda variável até aqueles que fazem desta prática sua principal fonte de renda. Entretanto, estas ferramentas geralmente possuem um custo mensal de utilização e não fornecem uma flexibilidade de customização suficiente para a exploração de estratégias mais refinadas de investimento, como o *trading* algorítmico, que consiste, de forma resumida, no emprego de

[1] Perfil pessoas físicas: B3 (2023c)
[2] A descoberta da bolsa pelo investidor brasileiro: B3 (2020)

software para automatizar o processo de investimento e a tomada de decisão, buscando a maximização dos lucros.

A execução deste tipo de software envolve o consumo de bases de dados (reais ou fictícias) que permitam simular, com determinado nível de precisão, a dinâmica de preços dos ativos financeiros ao longo do tempo. No contexto de estratégias de investimento algorítmicas, tem-se o costume de realizar simulações *ex-post*, isto é, utilizar dados reais do passado para analisar a performance de um modelo em um cenário específico, otimizando-o o e preparando-o para um mercado real.

A literatura relacionada a este tema é bastante ampla, principalmente quando o assunto principal são técnicas e modelos de investimento baseadas em modelos de inteligência artificial. Entretanto, observa-se uma relativa escassez de trabalhos focados no desenvolvimento de ferramentas que permitam a execução e avaliação destes modelos, como simuladores baseados em mercados eletrônicos reais.

1.1 Objetivos

O objetivo deste trabalho é desenvolver um *software* de investimentos de propósito educacional que permita a simulação *ex-post* de estratégias de negociação baseadas em *trading* algorítmico, sobretudo aquelas que utilizam como premissa o *trading* de alta frequência (HFT, do inglês *High Frequency Trading*). O simulador deve ser capaz de reproduzir o comportamento do mercado em um momento do passado (função de *replay*) a partir de dados históricos do mundo real fornecidos pelo servidor público de arquivos da B3. Espera-se obter uma ferramenta precisa (em relação à semelhança dos dados simulados em comparação com resultados do mundo real), performática (em termos computacionais, e sobretudo para estratégias de curto prazo) e flexível, no sentido que dê liberdade suficiente ao usuário final, permitindo a criação e personalização de diferentes estratégias de investimento.

1.2 Resultados

O *software* desenvolvido cumpriu de forma satisfatória com os objetivos estabelecidos, e a análise dos dados realizadas ao fim do documento destaca a semelhança entre as dinâmicas de mercado do mundo real e a simulada através dos dados. Além disso, a simulação a partir de dados reais históricos permitiu a extração de informações valiosas relacionadas ao mercado estudado, como os mapas de calor do livro de ofertas, que servem como referência para o entendimento das posições de mercado ao longo do dia para um determinado ativo financeiro. Ademais que, em termos de performance, foi possível simular grandes quantidades de dados em pequenos intervalos de tempo para ativos financeiros com menor liquidez. Mais adiante, será mostrado que os principais fatores limitantes de

performance do simulador estão relacionados com a quantidade de informação mantida em memória durante a execução do programa, que inevitavelmente devem ser guardadas, a fim de reproduzir o estado do mercado em cada instante de tempo.

1.3 Estrutura do documento

A estrutura restante deste documento está dividida em quatro capítulos. O Capítulo 2 fornece uma base teórica sobre mercados financeiros e bolsas de valores eletrônicas, apresentando conceitos essenciais para o entendimento das decisões tomadas durante o desenvolvimento do *software*. O Capítulo 3 trata da etapa de desenvolvimento do *software*, apresentando a arquitetura do projeto e detalhando os principais componentes que compõem a aplicação. No Capítulo 4 são apresentados os resultados obtidos através de simulações executadas sobre ativos de interesse, e as conclusões que podem ser tomadas a partir destas simulações. Por fim, o Capítulo 5 fornece uma conclusão para o presente trabalho realizado, apresentando alguns *insights* relacionados à realização de trabalhos futuros que venham a complementar este projeto.

2 Fundamentação teórica

O capítulo 2 apresenta algumas das definições base necessárias para o desenvolvimento do texto. Nas seções seguintes serão apresentados conceitos importantes relacionados aos mercados de ações (em especial, da bolsa de valores brasileira), sistemas de *trading* financeiro e alguns detalhes técnicos necessários para a implementação do ambiente de simulação. Os conceitos de *trading* algorítimico e *high frequency trading* serão brevemente discutidos, e em seguida serão apresentados os padrões de comunicação frequentemente utilizados em sistemas de *trading* eletrônico.

2.1 Bolsas de valores

Bolsas de valores são mercados organizados onde títulos financeiros são negociados de acordo com regras e regulamentações específicas, permitindo com que investidores realizem operações de compra e venda visando o lucro. Segundo (Smith, 2004) o primeiro exemplo de mercado de ações data do século II AC na antiga república de Roma, onde entidades legais conhecidas como *publicani*[1] tinham suas ações (*partes*) negociadas em uma região próxima ao templo de Castor. Após séculos de mudanças e evolução no mercado de capital, surgiram as primeiras bolsas nos moldes das existentes atualmente, com destaque para a bolsa de valores de Nova York (NYSE) criada no fim do século XVIII e que hoje é considerada a principal bolsa de valores do mundo. Antes do surgimento e disseminação dos computadores modernos, o processo de negociação de ações nas bolsas envolvia a reunião presencial dos investidores em uma área designada, onde a partir de um método conhecido como *open outcry* (pregão aberto) as ações eram negociadas. Similarmente a um leilão, os investidores competiam por ordens de compra e venda, utilizando majoritariamente a comunicação verbal e de sinais feitos com a mão. Com o surgimento e advento dos computadores modernos, este processo ficou cada vez mais obsoleto, dando lugar aos sistemas eletrônicos de negociação de ordens, que segundo Scott (2023b) ganharam bastante notoriedade no início da década de 1990, com a criação do primeiro sistema eletrônico de *trading* ao nível mundial, na Bolsa de valores de Chicago.

No Brasil, o principal mercado de negociação de ações é a B3, empresa que centraliza o mercado de ações nacional, estando entre as maiores bolsas de valores do mundo. A estrutura atual da bolsa é o resultado da união em 2008 de duas outras bolsas brasileiras: a Bolsa de Mercadorias e Futuros (BM&F), estabelecida em 1986, e a Bovespa, fundada no

[1] *Publicani*: Antigo empreiteiro público romano, que erguia ou mantinha prédios públicos, fornecia suprimentos para exércitos no exterior ou recolhia certos impostos, especialmente aqueles que forneciam quantias variáveis de receita para o estado como dízimos e tarifas alfandegárias.Britannica (2023).

final do século XIX.[2] De acordo com o próprio site da B3, a entidade é definida da seguinte forma:

> A B3 é uma das principais empresas de infraestrutura de mercado financeiro no mundo, com atuação em ambiente de bolsa e de balcão. [...] As atividades incluem criação e administração de sistemas de negociação, compensação, liquidação, depósito e registro para todas as principais classes de ativos, desde ações e títulos de renda fixa corporativa até derivativos de moedas, operações estruturadas e taxas de juro e de commodities. Institucional… (2023, Site B3)

Em termos de infraestrutura de negociações, a B3 conta hoje com a plataforma PUMA (Plataforma Unificada Multiativos) para a negociação eletrônica de ativos. A plataforma foi desenvolvida em parceria com a Bolsa Mercantil de Chicago (CME), sendo lançado para o público no primeiro semestre de 2013. A partir da plataforma PUMA, investidores e entidades financeiras podem operar sobre diferentes classes de ativos com segurança e eficiência.

No ambiente de negociações da B3, diferentes tipos de ativos podem ser comercializados, e cada classe de ativos possui diferentes riscos e margens de lucratividade. No mercado de renda variável, o principal tipo de ativo negociado são as ações de empresas listadas na bolsa. A listagem das ações de uma empresa na bolsa de valores passa primeiro por um processo de IPO (*Initial Public Offering*), regulamentado pela comissão de valores mobiliários (CVM), e tem como principal objetivo o levantamento de capital para aquela empresa. A partir desta abertura para o mercado, é possível que outras empresas e investidores individuais negociem pequenas frações deste ativo, visando lucrar ultimamente com estas operações. Outra classe de ativos bastante popular no mercado de ações brasileiro são os fundos imobiliários (FIIs), onde diversas pessoas aplicam seus recursos financeiros em um fundo de investimento fechado, que aplica estes recursos em ativos relacionados ao mercado imobiliário. Nesta modalidade de investimento os investidores recebem rendimentos proporcionais ao lucro sobre contratos de aluguel e venda de imóveis. Outro tipo de ativo bastante popular são os ETFs, que consistem em fundos de investimento que replicam índices do mercado, podendo estar relacionados a ações, criptomoedas ou até mesmo títulos de renda fixa. Além disso, existe também o mercado de derivativos B3 (2023a, B3 Derivativos), que são títulos financeiros cujo valor deriva de outro ativo, como taxas de juros, moedas, *commodities*, ações, etc.

Outro conceito relevante relacionado às bolsas de valores é o de participantes de mercado. Em Cartea, Jaimungal e Penalva (2015, seção 1.2), os autores ressaltam que apesar das diferenças de perfil, em última instância, a motivação por trás dos diferentes *players* do mercado é sempre a mesma:

[2] História do Mercado de Capitais: Historia… (2023).

> *Everyone's motivation is clear, they want to make money, but it is essential to consider what drives them to trade.* Cartea, Jaimungal e Penalva (2015, pg. 6)

Neste contexto, diferentes perfis de *traders* devem ser considerados, a fim de entender o impacto de cada um sobre o mercado. Em uma primeira instância, podemos considerar os gerentes de empresas listadas na bolsa como participantes de mercado, já que são os responsáveis pela criação dos ativos que serão negociados e que em determinado momento podem participar ativamente do mercado a partir do aumento ou da diminuição da quantidade de ações disponíveis. Uma outra classe de participantes de mercado com bastante impacto sobre o mercado são os fundos de investimento, que muitas vezes gerenciam grandes quantidades de contratos, representado uma porcentagem significativa das negociações geradas em um ambiente de *trading*.

Outro aspecto a se levar em conta ao classificar os participantes de mercado além do volume de ofertas gerado é quanto às estratégias de negociação. Neste sentido, *traders* individuais (como *day traders*) e fundos *hedge* possuem em comum o fato de negociarem ativos a partir da identificação de oportunidades de mercado, geralmente atreladas a técnicas de análise gráfica, análise de notícias relacionadas ao mercado, identificação de distorções nos preços, etc. Outra categoria de participantes de mercado é a de *traders* fundamentalistas. Estes agentes atuam com estratégias visando lucro a médio e longo prazo, a partir da análise de variáveis externas ao mercado, como o desempenho de uma empresa, tendência de crescimento de um setor do mercado, necessidade de rebalanceamento de carteiras e etc.

Em Cartea, Jaimungal e Penalva (2015), os autores defendem que todos estes agentes podem ser classificados nas seguintes categorias:

- *Fundamental traders*: *traders* que baseiam suas operações em uma análise de variáveis externas ao mercado, pautadas em fundamentos econômicos, como análise de performance e crescimento do setor de mercado.
- *Informed traders*: operam a partir do levantamento de informações não relacionadas diretamente aos preços de mercado, lucrando a partir da antecipação de aumentos ou diminuições nos preços.
- *Market makers*: *traders* profissionais, que atuam de forma reativa ao mercado, baseando suas estratégias em informações detalhadas do mercado, como histórico de preços, volume de negociações e tendências históricas.

A diferenciação dos participantes de mercado em um ambiente de negociação eletrônico nos permite classificar estratégias de *trading* algorítmico como uma parcela dos *market makers*, considerando que estes algoritmos atuam de forma reativa, a partir de informações retiradas diretamente do mercado.

2.2 Plataformas Eletrônicas de Negociação

O conceito de *trading* pode ser definido simplesmente como o processo de troca de posse de um determinado ativo entre um comprador e um vendedor. O tipo do ativo negociado pode ser bastante diverso, e ao longo da história pôde-se observar a troca de diferentes ativos, como, por exemplo, produtos agrícolas (grãos e sementes), metais preciosos, serviços especializados e títulos (ou contratos) que representem a posse de algo. Para facilitar estas trocas, foram criadas as bolsas, que como explicado na seção anterior, consistem basicamente em um mercado onde pessoas se reúnem para comprar e vender ativos específicos. Apesar de possuir uma definição simples, o processo de *trading*, entretanto, pode ser bastante complexo, já que envolve regras e regulamentações que garantem com que as negociações ocorram de forma justa e as transferências de posse sejam realizadas de maneira segura e transparente. Naturalmente, quanto maior for a quantidade de participantes do mercado, maior será a complexidade envolvida no processo de *matching* de valores e em manter um mercado justo para todos.

Como forma de resolver este problema, as primeiras bolsas de valores utilizavam de um método conhecido como pregão aberto (*open outrcry*) para padronizar o fluxo de negociações e garantir um ambiente mais confiável. A estrutura por trás deste processo, que foi utilizado por várias décadas, envolvia um longo processo manual onde muitas vezes eram necessárias centenas de pessoas para garantir com que as negociações fossem registradas, executadas, e posteriormente, que os contratos fossem validados. A Figura 2.1 ilustra este processo na forma de diagrama.

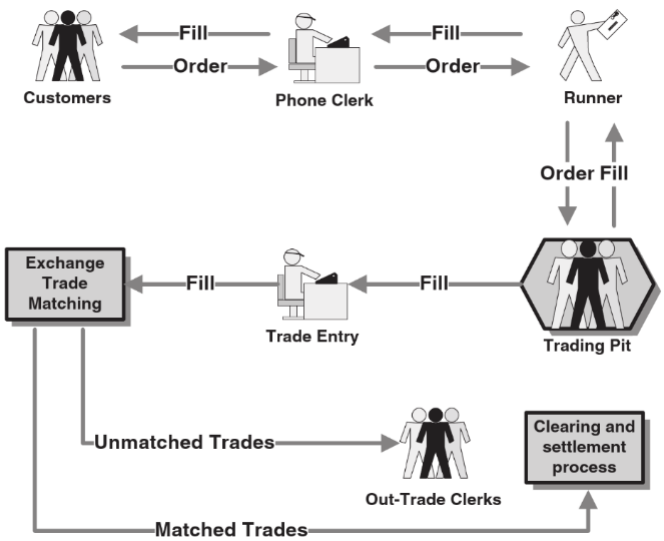

Figura 2.1 – *Open Outcry*: Modelo de *floor-trading*

Fonte: Gorham e Singh (2009, Figura 1.2)

Como pode ser observado, o processo envolve uma longa cadeia de execução, tornando-o relativamente lento (principalmente em comparação com o *trading* eletrônico) e complicado. Além disso, por envolver a interação humana em diferentes níveis, ocorriam muitos erros na passagem de informação, o que gerava diversos problemas envolvendo os preços negociados e as quantidades trocadas. Ao fim do dia, todas as negociações registradas passavam pelos processos manuais de *clearing* e *settlement*, onde eram revisadas e em seguida, ocorria a transferência de posse dos recursos financeiros.[3]

Com os recorrentes avanços tecnológicos do último século, sobretudo nas décadas de 1980 e 1990, quando os computadores pessoais se tornaram cada vez mais populares, tornou-se inevitável a transição dos sistemas de negociação para o modelo eletrônico. Dentre diversos outros motivos, essa mudança foi motivada principalmente pelo aumento na eficiência das operações, redução de custos operacionais, transparência, segurança e a possibilidade do acesso global aos mercados.

Em sistemas eletrônicos de negociação existem basicamente dois tipos de ordens: ordens à mercado e ordens limite. Ordens à mercado são aquelas que agridem diretamente o livro de ofertas, visando executar o *trade* imediatamente. O envio de uma ordem à mercado indica que o *trader* deseja negociar um ativo no melhor preço de mercado existente no instante em que a oferta será processada. Para ofertas de compra, isso significar comprar as ofertas de venda com o menor valor no mercado. Analogamente, para ofertas de venda, busca-se realizar uma negociação com os compradores que oferecem o maior preço no mercado. Dessa forma, o trade é realizado praticamente imediatamente após o envio da ordem de compra/venda. Já as ordens limite são aquelas onde deseja-se negociar uma quantidade do ativo a um preço específico. Este preço (limite) delimita se a ordem será executada imediatamente ou não. Para ofertas de compra, serão realizados negociações com ofertas de venda que estão listadas com um preço menor ou igual ao preço limite. De maneira similar, ofertas de venda com um valor limite serão executadas a um preço mínimo. Na maioria das vezes este tipo de ordem não será executado imediatamente, já que os preços de compra/venda possuem valores inferiores (ou superiores, no caso de venda) aos melhores preços do mercado. Estas ordens passam então a preencher o livro de ofertas, até que sejam executadas (preenchidas) ou canceladas.

O gerenciamento e a execução das ordens é realizado pelo *Limit Order Book* (LOB) e pela *engine* de *matching*, respectivamente. A partir destas duas estruturas é possível gerenciar e armazenar a entrada e saída de ofertas, e também realizar as negociações a partir de um conjunto específico de regras.

[3] Os processos de *clearing* e *settlement* ocorrem logo após o *trade* ser realizado. *Clearing* ocorre primeiramente, onde os termos de negociação são checados mais uma vez. *Settlement* é a última etapa do trade, onde a transferência de posse e dos recursos financeiros é realizada. Eisenhardt (2023)

2.2.1 Engine de Matching

O *matching* (ou correspondência) das ordens é o processo pelo qual uma *engine* de *matching* faz o pareamento entre ofertas de compra e venda, gerando novas negociações. Duas ofertas serão compatíveis quando o preço limite (máximo) de compra for maior ou igual ao valor limite (mínimo) de venda. Esse processo é realizado de maneira automática por um algoritmo inserido em sistemas de negociação eletrônicos, e a velocidade com que o sistema é capaz de realizar o pareamento das ofertas pode influenciar diretamente no preço executado, gerando diminuição nos lucros dos investidores. Em mercados eletrônicos, o modelo mais comum de pareamento de ordens é baseado na prioridade temporal, definindo uma fila de ofertas organizada por preço e tempo em uma estrutura conhecida como FIFO (*First in, first out*).

No modelo FIFO a prioridade é dada pelo tempo, onde ofertas de compra mais antigas e com o maior preço têm prioridade sobre ofertas subsequentes. Analogamente, ofertas de venda mais antigas e com menor preço possuem prioridade sobre ofertas de venda com preços maiores.

Outro modelo de *matching* de ordens é o *prorata*, que consiste na distribuição proporcional das negociações entre os participantes de mercado, levando em conta a quantidade de ordens e os preços oferecidos. Neste modelo, ordens com o mesmo preço e quantidades diferentes que não podem ser totalmente preenchidas são parcialmente executadas, de forma proporcional. O objetivo principal é garantir uma distribuição igualitária das ordens entre os participantes, sem levar em conta a ordem em que as ofertas foram recebidas.[4]

Na bolsa de valores brasileira (B3) utiliza-se o modelo de prioridade por tempo para realizar o casamento entre as ordens junto a um LOB que gerencia a entrada e saída de ofertas.

2.2.2 *Limit Order Book*

Como citado nas seções anteriores, o gerenciamento da entrada e saída de ofertas é realizado pelo livro de ofertas limite. O estado do livro de ofertas em um determinado instante de tempo t é definido pelas ofertas limite não executadas que chegaram ao mercado até o instante t. O preço de um ativo não é único, e para melhor representar o mercado, utiliza-se os preços de BID (maior preço de compra) e ASK (menor preço de venda). Quanto maior for o número de ofertas limite, maior será a liquidez do mercado, e consequentemente maiores serão as chances de ordens à mercado serem executadas a bons preços.

Neste sentido, pode-se dizer que ordens limite geram liquidez, enquanto as ordens à mercado consomem essa liquidez. A diferença entre os preços de BID e ASK define o *spread* do ativo financeiro, que representa a diferença entre os preços que os compradores estão

[4] Matching Orders: What They Are, How They Work, and Examples. Scott (2023a)

dispostos a pagar e os preços que os vendedores estão dispostos a vender. Sendo assim, o tamanho do *spread* depende diretamente da liquidez do ativo. Geralmente, quanto menor o valor do *spread*, maior é será liquidez do ativo. Similarmente, quanto maior for o valor do *spread*, menor será a liquidez.

Outro parâmetro que ilustra o estado livro de ofertas é o preço médio do ativo, que consiste na média aritmética entre os preços de BID e ASK. Note que o valor de BID tende a ser sempre menor que o de ASK. Porém, em casos especiais é possível observar valores de BID iguais aos de ASK, o que gera um estado de *lock* do mercado por curtos períodos de tempo, até que novas ordens à mercado sejam criadas. A Equação 2.1 representa o cálculo deste valor.

$$Midprice(t) = \frac{P_{bid}(t) + P_{ask}(t)}{2} \tag{2.1}$$

Para ativos com menor liquidez, a definição do preço médio pode levar a uma visão distorcida do mercado, visto que o volume de ofertas nos preços de BID e ASK pode ser muito diferente. Neste sentido, outro parâmetro utilizado para definir o preço médio de um ativo no mercado financeiro é o de micro preço, onde o volume de ofertas disponíveis nos preços de BID e ASK são levados em consideração no cálculo. A Equação 2.2 representa o cálculo deste valor.

$$Microprice(t) = \frac{V_{bid}(t)P_{ask}(t) + V_{ask}(t)P_{bid}(t)}{V_{bid}(t) + V_{ask}(t)} \tag{2.2}$$

A Figura 2.2 ilustra na forma de um gráfico o LOB de um ativo fictício em um instante de tempo t. Note que no exemplo ilustrado o *spread* do ativo fictício é de apenas alguns centavos de dólar, o que indica uma alta liquidez.

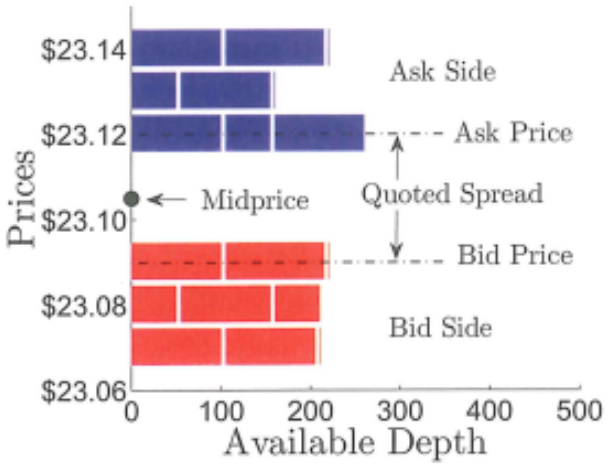

Figura 2.2 – *Limit Order Book* - Representação gráfica

Fonte: Cartea, Jaimungal e Penalva (2015, Figura 1.2)

Quando uma nova ordem limite é recebida, cabe ao LOB gerenciar a posição que esta oferta irá ocupar no livro. Primeiramente, comparam-se os preços das ofertas e em caso de empate utiliza-se o tempo como critério de desempate. A Figura 2.3 ilustra este comportamento para uma oferta limite com um preço superior ao BID atual, o que a coloca na primeira posição da fila de ofertas de compra.

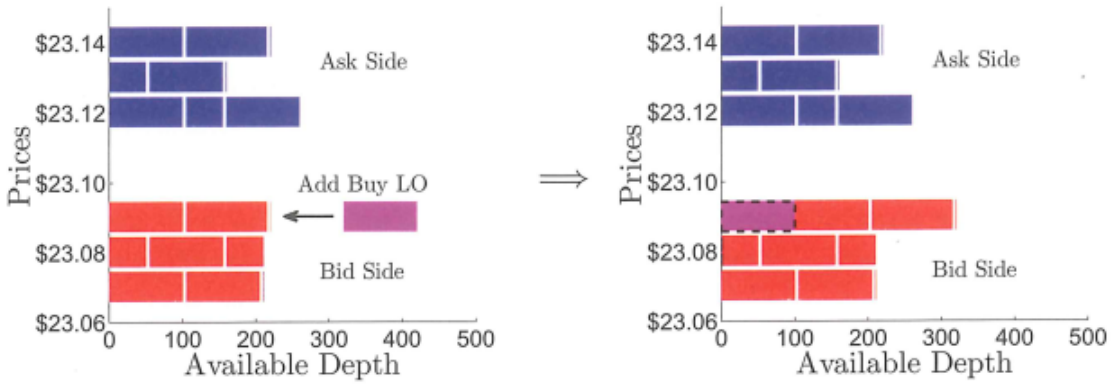

Figura 2.3 – Ordem limite chega ao LOB

Fonte: Cartea, Jaimungal e Penalva (2015, Figura 1.2)

Na Figura 2.4 é ilustrado o processo de *matching* de uma ordem à mercado com as ofertas limite que compõem o livro. Note que, caso a quantidade da oferta seja maior que o volume disponível nos preços de ASK/BID, os preços de negociação podem ser diferentes do esperado. Para contornar esta situação, certas bolsas contam com tipos específicos de ordens à mercado (e.g. *Immediate-or-Cancel*) que impedem com que a ordem seja executada a um preço diferente dos valores de ASK/BID do livro.

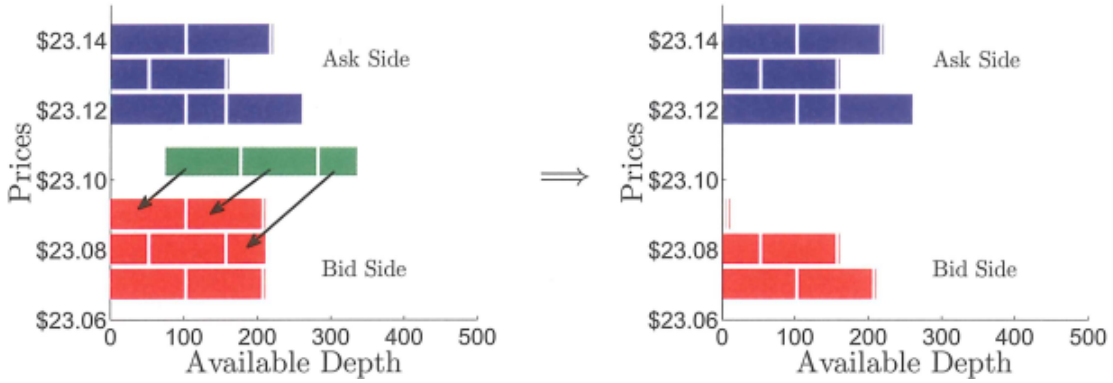

Figura 2.4 – Ordem à mercado chega ao LOB

Fonte: Cartea, Jaimungal e Penalva (2015, Figura 1.3)

2.3 Trading algorítmico

De acordo com Chen (2022), *trading* algorítmico consiste no processo computacional de executar ordens utilizando instruções de negociação automatizadas e pré-programadas, levando em conta variáveis como preço, *timing* e volume. O *trading* algorítmico faz uso de fórmulas complexas combinadas com modelos matemáticos e supervisão humana, para tomar decisões de compra e venda de títulos financeiros em uma bolsa de valores.

Com a evolução e o barateamento das tecnologias computacionais iniciada na última década do século passado, os mercados financeiros passaram a ser cada vez mais globais e complexos. Esta mudança de paradigma gerou um aumento significativo no número de participantes de mercado que utilizam sistemas automáticos de *trading* como forma de maximizar seus lucros. Esta modalidade de investimento consiste no uso de algoritmos que buscam por anomalias temporárias nos preços de ativos através de análises históricas de dados e padrões estatísticos, gerando ordens de execução em momentos precisos.

Dentre as principais vantagens relacionadas ao uso deste tipo de sistema, estão a velocidade com que os computadores conseguem tomar decisões (humanamente impossíveis de serem replicadas), redução de riscos relacionados à natureza humana, incluindo fatores psicológicos e emocionais e a capacidade destes sistemas de processar grandes quantidades de informações de mercado de forma simultânea, garantindo uma melhor tomada de decisão. Como *trade-off*, obtém-se um modelo altamente dependente de tecnologia e infraestrutura de ponta, o que aumenta significativamente o nível de investimento necessário para criar, testar e pôr a prova novas estratégias de *trading*. Além disso, o tempo de desenvolvimento de novas estratégias também é um fator relevante, visto que todo o processo de desenvolvimento e implementação do software pode levar vários meses ou anos. O processo de *trading* algorítmico envolve basicamente três etapas, divididas em: análise pré-trade, geração de sinal de *trading* e execução do *trading*. A Figura 2.5 ilustra os componentes de sistema relacionados a este processo.

A análise pré-trade é o caso mais comum do uso de algoritmos de *trading*. Consiste basicamente na utilização (em conjunto) de modelos de predição de preços de ativos do mercado (*alpha model*), modelos de gerenciamento de riscos de mercado associados a um ativo financeiro (*risk model*) e modelos de análise dos custos de transação associados à negociação de instrumentos financeiros. A etapa de geração de sinais de *trading*, por sua vez, consiste na avaliação dos resultados obtidos na etapa de pré-trade e em uma posterior tomada de decisão relacionada à construção do portfólio do investidor, definindo quais ativos e em quais quantidades deverão ser realizadas as negociações. A execução do *trading* é feita logo após a geração dos sinais de *trading*, e um modelo específico de otimização define o momento em que as negociações devem ocorrer, assim como quais tipos de ordens deverão

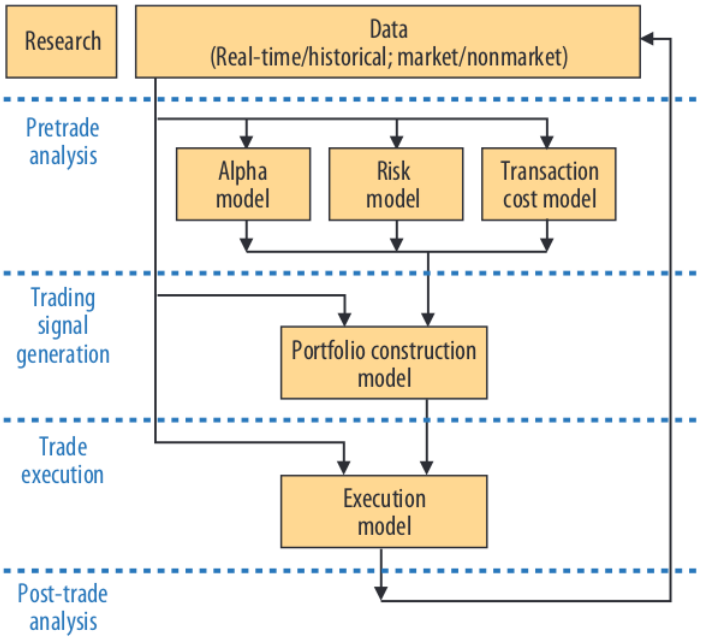

Figura 2.5 – Componentes de um sistema de *trading* algorítmico

Fonte: Nuti *et al.* (2011, Figura A)

ser geradas (ordens limite ou a mercado) e em quais mercados.[5]

Ao fim destas três etapas, o modelo utiliza as informações geradas até o momento da execução das ordens para realimentar a base de dados utilizada nas etapas anteriores.

2.3.1 High Frequency Trading

A partir da definição de *trading* algorítmico é possível introduzir o conceito de *High Frequency Trading* (HFT), que consiste em uma subclasse de sistemas contidos no conjunto de estratégias de *trading* algorítmico.

Por tratar-se de um tema relativamente recente, porém com grande variedade de estudos relacionados, existem múltiplas definições para o termo. Em (Aldridge, 2013), algumas destas diferentes definições são apresentadas, e de maneira geral, todas elas convergem para o mesmo conceito: HFT consiste na aplicação de tecnologias avançadas de análise de dados, tomada de decisão e acesso a mercados eletrônicos que visa o lucro sobre instrumentos financeiros a partir de múltiplas negociações em um curto intervalo de tempo. Apesar de tratar-se de uma das possíveis implementações do conceito de *trading* algorítmico, algumas características permitem diferenciar o HFT de sistemas genéricos de *trading* automatizado. De acordo com (Gomber *et al.*, 2011), as principais características que diferenciam o HFT de outros sistemas de *trading* são as seguintes:

[5] Definições dos componentes de um sistema de *trading*: Nuti *et al.* (2011, *Algorithmic Trading*)

- Elevado número de ordens: várias ordens de compra e venda são geradas de uma só vez pelo modelo de execução de ordens. Como o lucro por oferta é pequeno, busca-se multiplicar este valor a partir da execução de múltiplas ordens.
- Rápido cancelamento de ordens: necessário para a realização de pequenos lucros por negociação em um curto período de tempo.
- *Trading* proprietário: como a infraestrutura tecnológica por trás do HFT é extremamente robusta, utiliza-se capital proprietário para o desenvolvimento dos sistemas e para realizar a execução das ordens no mercado.
- Lucros na compra e na venda de ativos: a estratégia atua como uma intermediária no mercado, comprando e vendendo ativos simultaneamente.
- Posição praticamente zerada ao fim do dia: as estratégias em HFT são orientadas a movimentos de curto prazo, portanto, busca-se encerrar todas as posições até o fim do dia. Além disso, essa prática permite evitar riscos relacionados a eventos macroeconômicos, notícias inesperadas e custos relativos à manutenção da posição.
- Períodos de *holding* extremamente curtos: fluxo de geração de ordens (pré-trade, geração de sinal e execução de ordem) geralmente na casa de millisegundos.
- Margem de lucro baixa por *trade*: como as negociações ocorrem em curtos períodos de tempo, a oscilação nos preços é pequena, e, portanto, o lucro por negociação (que é proporcional à essa oscilação) também é pequeno.
- Requisito de latência baixa: toda a dinâmica do HFT baseia-se em uma alta velocidade de acesso aos mercados e troca de informações, e por conta disso é de extrema importância que seja mantida uma baixa latência de comunicação entre o sistema e o servidor do mercado.
- *Co-location*[6]: como forma de se diminuir a latência, a instalação física destes sistemas é feita próxima aos servidores do mercado. Dessa forma diminui-se a distância percorridas pelos dados durante a comunicação entre os sistemas, e consequentemente diminui-se o tempo de resposta.
- Necessidade de alta liquidez do ativo: Como o lucro por *trade* é pequeno, são necessárias várias negociações para se obter um lucro significativo, e portanto, é necessária uma alta liquidez do instrumento.

[6] *Co-location* refere-se à maneira como os equipamentos e recursos de TI são localizados ou instalados. Isso geralmente se refere aos recursos de hardware de rede pertencentes a uma organização, como servidores da Web ou de banco de dados, que estão localizados fora das proximidades das instalações da organização e "co-localizados" com o hardware de outra organização. Techopedia (2017). A bolsa de valores brasileira (B3), por exemplo, fornece um ambiente de *co-location* para empresas e investidores profissionais.

2.3.2 FIX/FAST

Além da evolução do hardware utilizado para a implementação eletrônica dos sistemas de *trading* modernos, foram necessárias adaptações nos modelos de comunicação e transmissão de dados para garantir uma maior eficiência destes sistemas, visando extrair o máximo de performance possível. Em sistemas de *trading* eletrônico, um dos principais fatores limitantes em relação à velocidade com que as mensagens são trafegadas é a forma com que os dados são codificados e transmitidos.

Para o tráfego de informações de mercado, utiliza-se o modelo UDP para enviar dados da bolsa para o usuário. No protocolo UDP a velocidade com que os dados são trafegados é alta, porém não há garantia de que todos os pacotes enviados serão entregues ao destinatário. Por essa razão, utiliza-se este protocolo apenas para o envio de dados menos sensíveis a erros, como dados de cotação de ativos, enviados com uma frequência relativamente alta, o que faz com que pacotes com informações erradas sejam corrigidos automaticamente nos próximos pacotes de dados recebidos.

No contexto de *algorithmic trading*, a estrutura com que estes dados são recebidos pelo algoritmo é de extrema importância, visto que é a partir destes dados que serão gerados sinais de *trading*. Dessa forma, é indispensável garantir certo grau de equivalência entre os formatos de dados utilizados em simulações (*backtesting*) e no ambiente de produção (execução do modelo em um ambiente real).

Para a transmissão de dados relativos à criação, execução e cancelamento de ordens, bem como outros dados sensíveis, utiliza-se o protocolo TCP/IP para comunicação. O protocolo TCP/IP possui uma estrutura de cabeçalhos que permite a identificação de erros nos pacotes recebidos, o que resulta em uma comunicação mais confiável entre servidor e cliente, ao mesmo tempo que gera uma redução significativa na velocidade com que os dados são transmitidos quando comparado ao protocolo UDP.

O formato dos dados transmitidos, por sua vez, possui uma estrutura específica, definida a partir de protocolos criados com o intuito de padronizar e aumentar a eficiência de comunicação entre sistemas de *trading* eletrônicos. A Figura 2.6 ilustra uma possível arquitetura de comunicação utilizando os protocolos UDP e TCP/IP para transmissão de dados em conjunto com o protocolo FIX (*Financial Information Exchange*) para a codificação de dados.

Como pode ser observado da imagem, o protocolo FIX é utilizado tanto para cotação de ativos quanto para transmissão de mensagens de negociação, geração de ordens, etc. As mensagens codificadas utilizando o protocolo FIX possuem campos de dados enumerados que comprimem as informações úteis no contexto da mensagem em uma estrutura específica. Dentre os campos codificados na mensagem, encontram-se informações relevantes para o contexto da execução, como versão do protocolo FIX, identificador temporal da mensagem

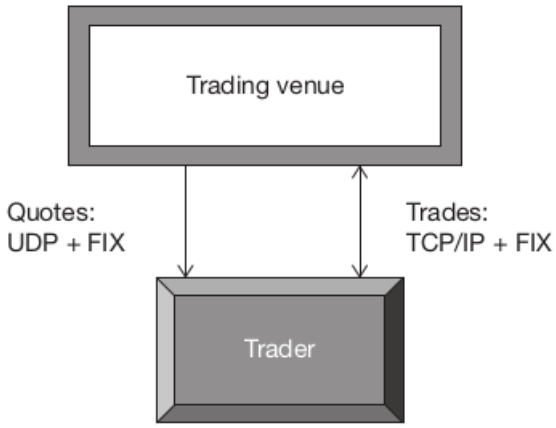

Figura 2.6 – Arquitetura de comunicação utilizando FIX

Fonte: Aldridge (2013, Figura 2.7)

(*timestamp*), moeda base, identificador do ativo (símbolo), preços de BID e ASK, etc.

Além do protocolo FIX, outros protocolos também são utilizados, como o ITCH e OUCH (Nasdaq). Nestes protocolos, a codificação dos dados é feita utilizando código binário, onde não há a necessidade da decodificação da mensagem pelo destinatário final. Por conta disso, têm-se uma comunicação mais rápida que a do protocolo FIX. Apesar disso, os protocolos ITCH e OUCH possuem limitações quando comparados ao FIX, sobretudo na complexidade do tipo das mensagens que podem ser transmitidas.

A B3 utiliza o protocolo FAST (*Fix Adapted for Streaming*) para a codificação de mensagens [7].O Protocolo FAST é desenvolvido e mantido pelo *FIX Trading Community's Market Data Optimization Working Group* e tem como objetivo possibilitar o uso eficiente da largura de banda em mensagens de alto volume sem causar sobrecarga significativa de processamento ou latência. O protocolo FAST foi desenvolvido como resultado de uma prova de conceito do *Market Data Optimization Working Group* para explorar vários métodos de representação de dados otimizados dentro do Protocolo FIX. [8]

[7] FIX/FAST *Market Data Messaging Specification*. B3 (2023b)
[8] FAST *Protocol*. Community (2016)

3 Desenvolvimento do simulador

O Capítulo 3 trata da etapa de desenvolvimento do simulador, e serão apresentadas as decisões técnicas referentes à concepção e construção do *software*. O sistema que deseja-se construir é algo próximo ao ambiente PUMA da B3, com algumas simplificações relacionadas aos protocolos de comunicação, visando acelerar o desenvolvimento.

3.1 Visão geral do projeto

Apesar da existência de uma ampla literatura no contexto de HFT, é escasso o conteúdo referente ao desenvolvimento de um ambiente de simulações que possibilite a execução e o estudo de algoritmos capazes de realizar operações de compra e venda em um curto período de tempo. Neste sentido, a construção da arquitetura do simulador e o seu posterior desenvolvimento foi realizado tendo como base experiências prévias em outros projetos de *software*, que apesar de serem direcionados para outras finalidades, compartilham de conceitos base que foram amplamente utilizados no projeto. Algumas dos conceitos técnicos mais importantes para a concepção do sistema envolvem a definição de estruturas de dados para lidar com a entrada e saída de dados, conceitos de programação com requisitos de tempo real (*threads*, mecanismos de sincronização, etc), compartilhamento de dados entre programas através de *sockets*, etc. De maneira simplificada, a aplicação consiste em dois programas principais, um cliente e um servidor, que comunicam-se por meio de um *socket*. Cabe ao servidor gerir a dinâmica do mercado, isto é, executar ordens de compra e venda e montar o livro de ofertas. Já o cliente é o responsável por gerenciar a conta do operador e executar a estratégia de negociação.

3.1.1 Escopo

Por tratar-se de uma simulação, entende-se que as funcionalidades implementadas devem ser uma aproximação do modelo real existente no ambiente da B3, e que estas funcionalidades devem ser tão próximas das reais quanto possível. As seguintes funcionalidades definem o conjunto dos principais requisitos que devem ser cumpridos pela aplicação:

- Integração com dados reais (Servidor): o servidor deve ser capaz de ler e processar arquivos de texto contendo dados históricos de execução do ambiente da B3. Estes dados devem ser lidos e processados sequencialmente, evitando ao máximo ocupar uma grande porção da memória RAM do computador onde a aplicação está sendo executada. Para isso, a fila de ofertas deve possuir um tamanho limite de 10000 ordens, e caso este limite seja ultrapassado, a leitura deve ser interrompida até que o processamento

de dados seja capaz de reduzir a fila ao tamanho máximo desejado. Cada oferta não processada ocupa 230 *bytes* de memória, sendo assim, para o pior dos casos, o livro de ofertas irá ocupar no máximo 2,3 *megabytes* da memória RAM.

- Montagem do livro de ofertas (Servidor): na rotina de processamento de uma nova ordem, o servidor deve ser capaz de atualizar o livro de ofertas caso se faça necessário. Novas ofertas devem ser ordenadas por preço e por tempo prioridade e atualizações de ordens podem refletir em perda de prioridade.

- Execução de ordens (Servidor): o servidor deve ser capaz de executar novas negociações, preenchendo parcial ou totalmente as ordens envolvidas na operação. Em caso de preenchimento parcial, a quantidade da ordem envolvida deverá ser atualizada, e em caso de preenchimento total, a ordem deverá ser removida do livro de ofertas. Esta regra vale tanto para a ordem que "agride" o livro de ofertas quanto às outras ordens envolvidas na negociação.

- Geração de eventos (Servidor): o servidor deve gerar eventos relativos à execução de ordens enviadas pelo cliente (preenchimento parcial e total), mudanças nos preços de BID e ASK, novas negociações e eventos relacionados a estatísticas de mercado, como o volume de ofertas por *range* de preço em um determinado instante da execução.

- Monitoramento de performance (Servidor): o servidor deve ser capaz de gerar e armazenar dados referentes à sua própria performance. Em especial, deve ser capaz de gerar dados relativos ao tempo de execução médio de um ciclo de ordem em um determinado instante de tempo, dado o tamanho (quantidade de ofertas) atual do livro de ofertas e da fila de ordens.

- Estratégia de *trading* customizada (Cliente): do lado do cliente, deve ser possível definir uma estratégia de *trading* algorítmica. Isto deve ser feito a partir de funções com nomes específicos que representam ações de reação a eventos gerados pelo simulador, como por exemplo, *onTrade*, *onBook*, *onTick*, etc.

- Envio de ordens (Cliente): o cliente deve ser capaz de enviar ordens de compra e venda à mercado e ordens limite ao servidor. Estas ordens deve ser colocadas no fim da fila de ordens, assim como as demais ordens lidas dos arquivos de dados.

- Gerenciamento da conta do operador (Cliente): o cliente deve ser capaz de gerir a conta do operador durante toda execução, armazenando dados relativos à estratégia (como perdas e ganhos) e a posição do operador no mercado. Além disso, o cliente deve validar operações de compra e venda, impedindo compras quando não há recursos monetários suficientes e cancelando vendas de ações que não estão presentes no portfólio do operador.

- Mínima interface gráfica (Cliente): o cliente deve fornecer uma interface gráfica para o envio de novas ordens e gerar gráficos relacionando os preços de negociação de um determinado ativo ao longo da execução.

A partir destes requisitos, têm-se a base funcional da aplicação desenvolvida.

3.1.2 Tecnologias utilizadas

Como dito anteriormente, a principal motivação por trás do desenvolvimento deste simulador é a necessidade de trabalhar em um cenário o mais próximo possível de um ambiente real. Além disso, a existência dos dados históricos de execução da B3, utilizados como *input* do programa, também foi uma grande fonte de motivação para a execução deste trabalho, já que com dados simulados dificilmente seria possível obter resultados próximos aos reais. Vale salientar que os arquivos de dados utilizados apresentam um volume considerável, ocupando cerca de 15 *gigabytes* de memória no total (para uma única data), o que aumenta o desafio de obter uma ambiente performático. Sendo assim, tendo em vista a natureza de tempo real de um ambiente de negociação de ações, é natural que um dos requisitos de projeto a ser cumprido seja a velocidade de execução de uma simulação, o que em outras palavras se traduz no tempo médio de execução de uma nova ordem. Em um ambiente real de negociações como o da bolsa de valores brasileira, existe uma complexa arquitetura de computação em nuvem que permite com que milhares de pessoas possam operar diariamente sobre diversos ativos de forma concorrente. Replicar este ambiente computacional foge do escopo deste trabalho, porém, a partir de otimizações de *software* e da escolha correta da arquitetura a ser seguida é possível alcançar resultados satisfatórios, como os apresentados no Capítulo 4.

Como o objetivo deste simulador é fornecer uma experiência próxima do cenário real, optou-se por desenvolver o *core* da aplicação (servidor) utilizando a linguagem de programação C++. Abaixo estão listados alguns dos principais fatores que influenciaram nesta escolha.

- Controle de memória: C++ permite a alocação e manipulação de memória em baixo nível, que no contexto deste simulador é imprescindível, já que a execução consiste, basicamente, em um *loop* de leitura e processamento de dados. Neste sentido, é necessário garantir que toda a memória que não esteja sendo utilizada seja liberada, prevenindo vazamentos de memória que poderiam afetar significativamente a performance do programa.
- Ecossistema: por ser uma linguagem bem consolidada no mercado e utilizada em diversos contextos, existem diversas bibliotecas e ferramentas que podem ser reapro-

veitadas durante o desenvolvimento do *software*, evitando retrabalhos desnecessários não relacionados com o objetivo do projeto.

- Portabilidade: é possível executar binários de projetos C++ nos principais sistemas operacionais e arquiteturas, facilitando futuras colaborações no projeto a partir de novas funcionalidades ou correção de *bugs*.
- Comunidade ativa: o C++ é uma das linguagens mais populares atualmente, ocupando a 9ª posição entre as linguagens mais utilizadas por profissionais no *rank* da plataforma *Stack Overflow* no ano de 2022.[1] Esta característica faz com que seja consideravelmente fácil encontrar soluções para eventuais problemas durante o desenvolvimento do código em fóruns ou sites especializados como o próprio *Stack Overflow*.
- Performance: por ser uma linguagem compilada, o C++ passa por diversas etapas de otimização durante o processo de transformação de código em binários pelo compilador. Além disso, por possuir características como gerência de memória em baixo nível, possibilitar o uso de ponteiros e possuir estruturas de dados extremamente eficientes, a linguagem mostra-se bastante eficiente em termos de tempo de execução de código.

O projeto inclui um módulo semelhante ao de um *homebroker*, que atua como um cliente do servidor, enviando ordens e recebendo eventos relacionados com o contexto da execução. Diferente do servidor, os requisitos funcionais deste cliente baseiam-sem principalmente em sua portabilidade e legibilidade, já que é neste módulo onde encontra-se toda a lógica relativa ao envio de ordens, gerência da conta do *trader* e definição da estratégia de negociação. Sendo assim, optou-se por utilizar a linguagem de programação *Python*, que compartilha de algumas características com o C++, como, por exemplo, seu ecossistema e comunidade, e que leva vantagem no quesito produtividade, oferecendo uma sintaxe mais simples e bibliotecas intuitivas para a plotagem de gráficos e manipulação de dados.

3.1.3 Metodologia de desenvolvimento

A metodologia de desenvolvimento de *software* em espiral foi escolhida como guia para a implementação do código do simulador. De forma resumida, esta metodologia permite com que o *software* sempre possua uma versão funcional, ao mesmo tempo que é melhorado e incrementado ao longo de iterações (ou ciclos). Por se tratar de um projeto realizado por uma única pessoa, não houve necessidade de criação de quadro de tarefas ou divisão de funcionalidades. Ao invés disso, pequenas alterações foram sendo feitas ao longo das iterações que duraram na maioria das vezes o período de uma semana. Como ferramenta de versionamento de código foi utilizado o *git*, que permitiu com que novas funcionalidades fossem trabalhadas em paralelo à versão funcional atual a partir da estrutura de *branchs*. Dessa

[1] Stack Overflow Developer Survey 2022: Overflow (2022)

forma, novas versões foram testadas e validadas antes de serem incorporadas à base principal de código, permitindo um fluxo de desenvolvimento contínuo pautado em funcionalidades.

3.2 Dados para simulação

Ao desenvolver um ambiente de simulação, independente da natureza do processo estudado, espera-se obter certo grau de fidelidade dos resultados em relação a dados reais que representem o processo. No mundo de finanças quantitativas, um conceito bastante utilizado é o de *backtesting*, que pode ser definido como se segue:

> Backtesting is the general method for seeing how well a strategy or model would have done ex-post. Backtesting assesses the viability of a trading strategy by discovering how it would play out using historical data. If backtesting works, traders and analysts may have the confidence to employ it going forward. Chen (2023).

Desta definição percebe-se a importância de lidar com dados reais ao trabalhar com simulações de mercado. Além de viabilizar possíveis otimizações da estratégia de *trading* antes de aplicá-la em um mercado real, é possível ter um certo grau de confiança nos resultados obtidos, já que as simulações foram realizados sobre dados reais.

Neste simulador, todos os testes realizados serão feitos tendo como base dados históricos (públicos) fornecidos pelo servidor de transferência de arquivos da B3. O conjunto de dados utilizados refere-se a todo o histórico de execução do servidor PUMA da bolsa de valores brasileira ao longo de um dia inteiro. Nestes arquivos estão registradas todas as ordens de compra e venda recebidas pela B3, bem como o histórico de execução de operações (preenchimento de ordens, cancelamento, negociação, etc) realizado pelo ambiente de *trading*. Os arquivos em questão são do tipo texto e possuem um padrão de organização das informações que permite com que um simples *parser* dentro do programa seja capaz de extrair informações extremamente úteis. A partir deste histórico de execuções, busca-se recriar a dinâmica do livro de ofertas e das negociações que ocorreram no mercado na data específica em que os dados foram salvos, e a partir disso, testar estratégias de *trading*.

3.3 Arquitetura de Software

De maneira simplificada, a arquitetura do simulador resume-se à dois processos diferentes comunicando-se através de um *socket* TCP, e trocando dados apenas quando necessário. Diferente de uma estrutura *client-server* convencional, como as encontradas em aplicações *web*, o servidor não depende de receber requisições do cliente para enviar dados, de modo que uma simulação completa pode ser executada sem haver interações na direção do cliente-servidor. Esta dinâmica foi definida com um objetivo simples: fazer do servidor

um ambiente isolado, capaz de executar o *replay* das negociações de um ativo na B3 em um determinado momento do tempo. Uma vez que este objetivo fosse concluído, seria então possível incluir um cliente (neste contexto, representando um *homebroker*) que pudesse participar (e de certa forma alterar) da dinâmica das negociações e flutuações do livro de ofertas durante a execução. Sendo assim, têm-se de certa forma uma estrutura "invertida" do modelo cliente servidor, onde o cliente recebe informações relativas à dinâmica de mercado, processa estes dados e em seguida decide se deve responder a essa requisição, conforme a estratégia de *trading* definida em sua estrutura.

A Figura 3.7 ilustra esta arquitetura, destacando, para cada componente, as camadas de lógica que relacionam as entidades desenvolvidas em código.

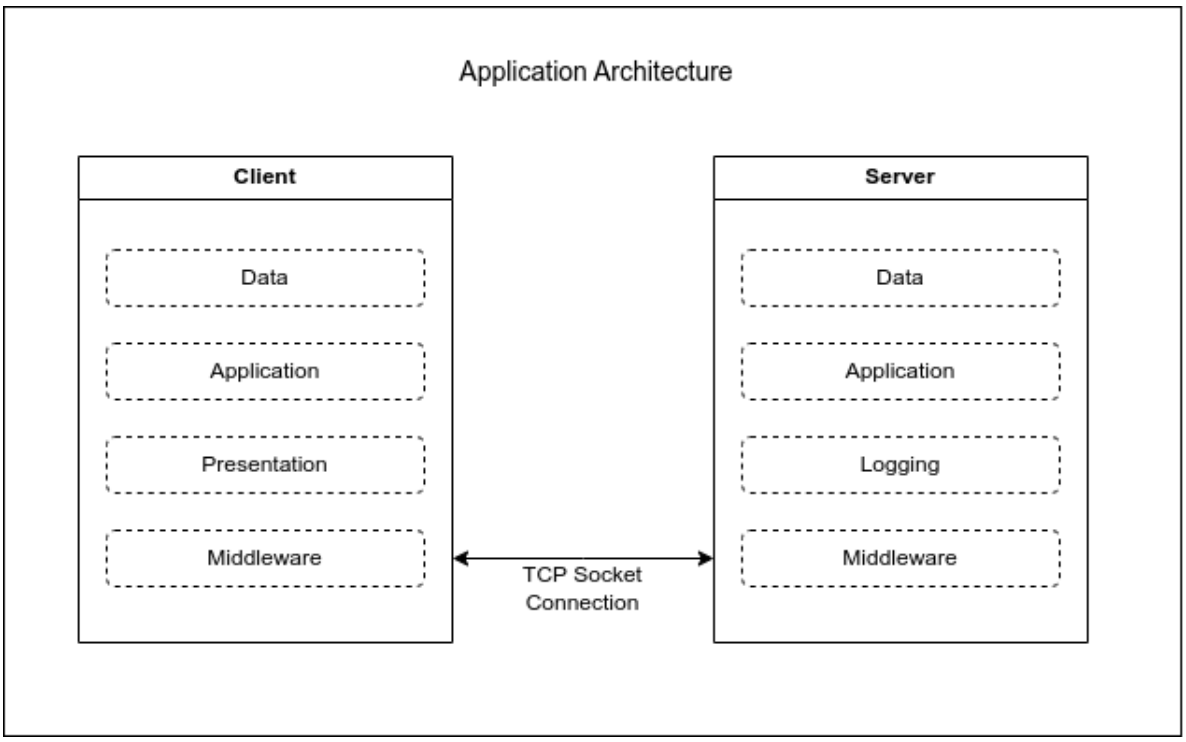

Figura 3.7 – Arquitetura simplificada do simulador

Fonte: Produzido pelo autor

Nas próximas seções será explorada de maneira detalhada toda a arquitetura envolvendo tanto o cliente quanto o servidor da aplicação.

3.4 Servidor

A Figura 3.8 ilustra o conjunto de componentes e estruturas de dados envolvidas na dinâmica de execução do servidor. Os principais serviços que compõe a aplicação foram agrupados por camada, e para cada camada estão ilustradas os serviços e as entidades envolvidas no fluxo de execução da simulação.

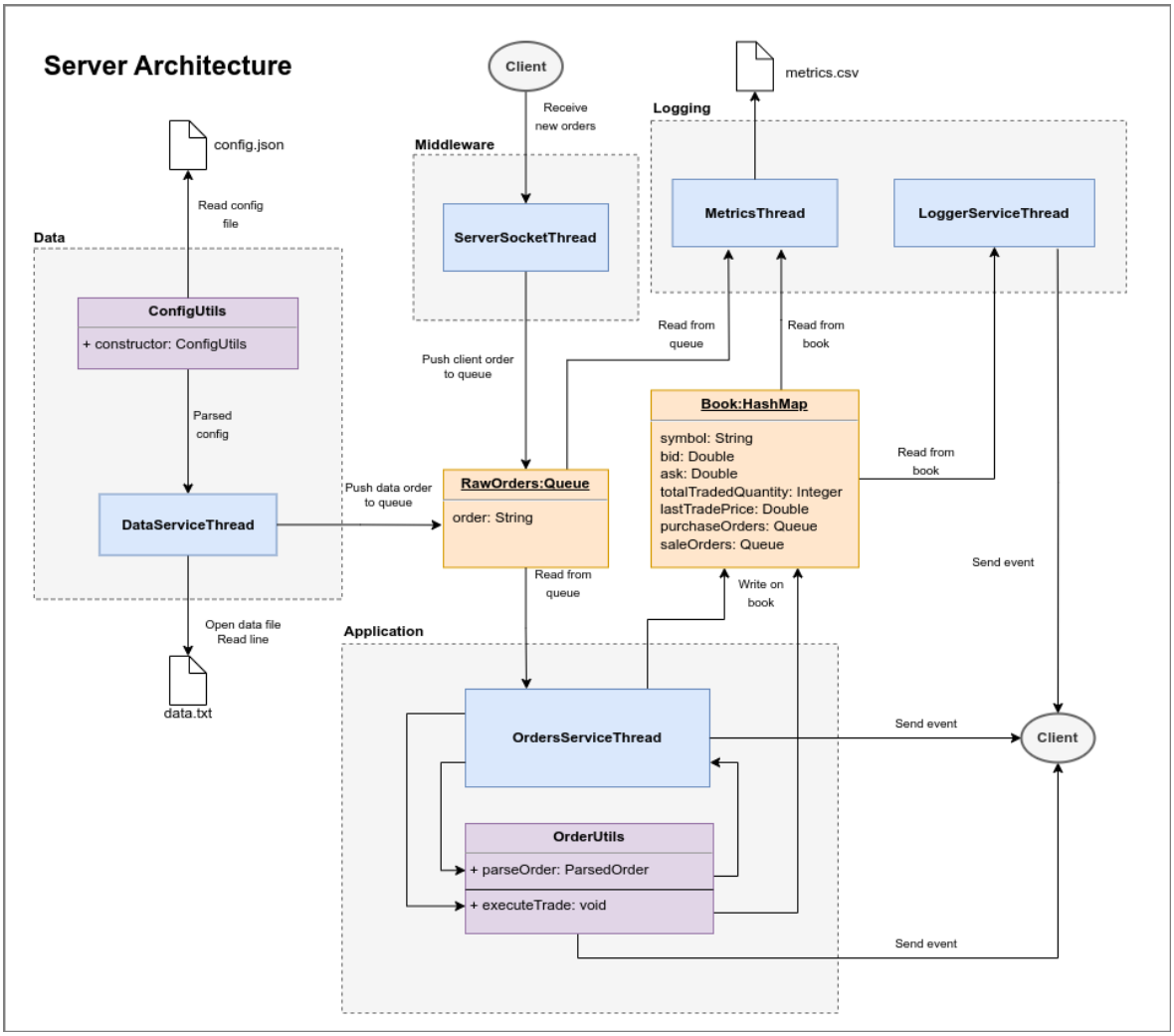

Figura 3.8 – Arquitetura do servidor

Fonte: Produzido pelo autor

É no servidor onde encontra-se toda a lógica responsável por acessar, ler e armazenar os dados históricos fornecidos pelo servidor FTP da B3. Quatro camadas distintas trabalham concorrentemente com responsabilidades bem estabelecidas. Na camada de dados (*Data*), encontra-se toda a lógica responsável pelo acesso e obtenção dos dados necessários para a simulação. É nela onde, a partir das definições feitas no arquivo de configuração, é feito o acesso e leitura do arquivo alvo, que é lido linha a linha, com um intervalo de tempo da ordem de millisegundos. Dessa forma, permite-se com que haja uma dinâmica de leitura e processamento de dados, evitando-se um *overflow* de memória desnecessário. A próxima camada é a de aplicação (*Application*), responsável pelo processamento dos dados que compõe a fila de ordens de execução. É nela onde ocorre todo o ciclo de vida de uma ordem, partindo do seu formato inicial, uma *string* não processada, até sua adição ao livro de ofertas e seu posterior preenchimento.

Paralelamente a estas duas camadas, encontram-se as camadas de comunicação

(aqui chamada de *Middleware*) e de registro de métricas do sistema (*Logging*). A camada de *Middleware* exerce papel central na comunicação com o cliente. É nesta camada onde é feita a criação do canal bidirecional de comunicação entre o servidor o cliente, permitindo o envio e o recebimento de dados entre as partes. Já na camada de *Logging* ocorre a criação e o armazenamento de informações referentes à performance do simulador (métricas de execução) por meios de arquivos do tipo CSV (*Comma-separated values*). Dois serviços distintos são executados simultaneamente nesta camada, um realizando o envio de informações referentes à execução ao cliente em intervalos de tempo igualmente espaçados e o outro escrevendo em um arquivo de métricas de execução. Ambos os serviços possuem a mesma finalidade: fornecer dados de execução que possam ser processados posteriormente, gerando *insights* em relação à performance do simulador e ao conjunto de dados simulados.

3.4.1 Mecanismos de Sincronização

Por possuir uma arquitetura majoritariamente pautada em programação concorrente, faz-se necessário o uso de primitivas de sincronização voltadas para a resolução de conflitos que surgem da necessidade de acessar recursos compartilhados entre múltiplas *threads*. Para este simulador, optou-se por utilizar uma estrutura de semáforo como forma de garantir o gerenciamento de recursos entre as seções de código que competem por uma variável compartilhada. Um exemplo desta utilização é facilmente observado na dinâmica de operação das camadas de dados e de aplicação, onde a camada de dados tenta, a cada iteração do seu *loop* de execução, adicionar novas ordens à fila de ordens, ao mesmo tempo que compete com a camada de aplicação, que busca consumir e remover ordens dessa mesma fila. A ordem de execução destas seções críticas é determinada pela estrutura do semáforo, bloqueando a execução da seção crítica de uma das *threads* ao mesmo tempo que garante o acesso exclusivo da outra. Dessa forma, mesmo com múltiplos trechos de código tentando acessar o mesmo recurso simultaneamente, mantém-se a integridade da simulação.

A simulação conta também como uma entidade global que representa um relógio, acessada por praticamente todos os componentes do servidor. Nesta entidade são definidos os tempos fictícios e real da simulação, permitindo gerar intervalos consistentes entre as leituras de ordens na camada de dados e sincronizar estes intervalos com o tempo (fictício) da simulação, gerando o mínimo de distorções possível na dinâmica que relaciona os preços do livro de ofertas com o tempo de execução da simulação, garantindo uma maior acurácia entre os dados reais e os resultados obtidos através da simulação.

3.4.2 Implementação

A implementação do servidor segue uma lógica simples: executa-se um loop de leitura e processamento de ordens até que todos os dados sejam lidos e todas as ofertas processadas.

A Figura 3.9 ilustra o fluxo de execução de uma ordem, e representa cada iteração do *loop* principal.

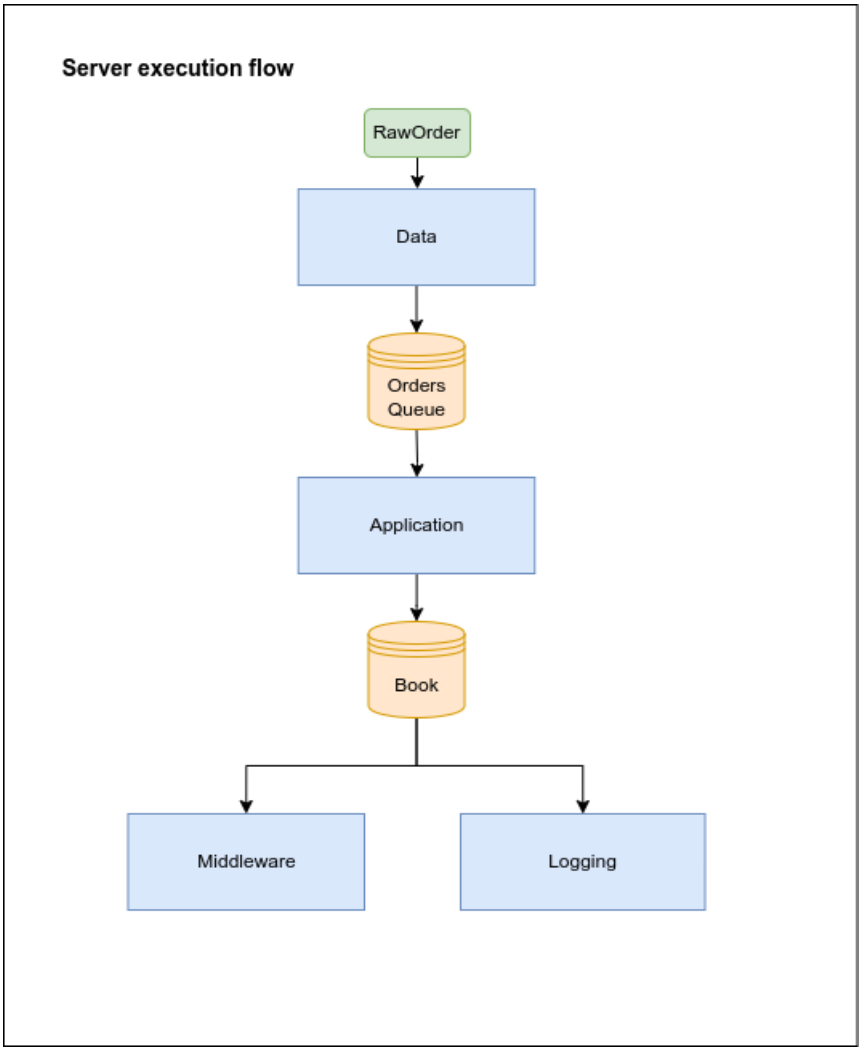

Figura 3.9 – Fluxo de execução do servidor

Fonte: Produzido pelo autor

Como pode ser observado da imagem, uma ordem de execução passa pela camada de dados, onde é adicionada à fila de ofertas, é processada pela camada de aplicação e em seguida passa por um processo de chaveamento, onde pode ser rejeitada, adicionada ao livro ou preenchida instantaneamente. Nas próximas seções serão explorados os detalhes técnicos relacionados à implementação em código das camadas apresentadas anteriormente.

3.4.2.1 Camada de Dados

Como citado anteriormente, a camada de dados é responsável pela leitura de novas ofertas do arquivo alvo e por preencher a fila de ofertas. Este processo inicia-se a partir da leitura de um arquivo de configuração do tipo JSON (*Javascript Object Notation*), que guarda informações como a data alvo da simulação, isto é, em que dia exato do calendário o arquivo

de dados foi gerado pelo servidor da B3, o diretório em que se encontra o arquivo na máquina local, o símbolo alvo, e a velocidade da simulação.

Os parâmetros de data, caminho relativo e símbolo alvo do arquivo de configuração são utilizados exclusivamente para encontrar e abrir o arquivo alvo no diretório especificado. Note que o símbolo alvo é um valor único, já que a versão atual do simulador só é capaz de trabalhar com um único ativo de cada vez. A decisão de trabalhar com apenas um ativo por vez foi tomada após algumas iterações do desenvolvimento do simulador. Inicialmente, a ideia era trabalhar com uma lista de ativos alvo, e uma das versões do simulador chegou a contar com esta funcionalidade, que foi rapidamente desfeita.

O motivo por trás dessa mudança se dá pela forma com que os dados históricos são organizados. Para cada tipo de ordem (compra, venda e negociação) existe um único arquivo, organizado alfabeticamente (por símbolo) e por tempo prioridade. Esta característica, aliada ao fato de que cada um destes arquivos ocupa em média 5 *gigabytes* de memória, gera um problema: digamos que, em determinado momento, o operador decide testar a mesma estratégia para dois ativos correlatos, o ativo A e o ativo Z. Considerando que os dados são organizadas primariamente por ordem alfabética, o simulador irá processar todos os ativos com o símbolo A antes de chegar às ordens do ativo Z, e, portanto, a estratégia será testada em sequência, e não paralelamente.

Uma forma simples, porém pouco eficaz de contornar este problema seria simplesmente ler e armazenar em memória todas as linhas de dados dos ativos de interesse e em sequência ordenar todas as ofertas antes que a aplicação começasse a consumi-las. Porém, como é de se esperar, esta ideia é extremamente ineficiente para ativos com grandes volumes de ofertas, já que, além de carregar uma enorme quantidade de dados em memória RAM, o simulador seria responsável também por analisar cada linha, extrair informações como símbolo e tempo prioridade e em sequência ordenar todos os dados na memória.

Na prática, a execução teria um início extramente lento, e a depender do ativo poderia exceder a quantidade de memória RAM disponível, já que, como veremos na seção de resultados, para o ativo com maior liquidez, existem cerca de 25 milhões de ofertas de compra e venda. Para referência, cada ordem de execução nos arquivos de dados possui exatamente 230 caracteres (230 bytes), ou seja, no pior dos casos, seriam necessários aproximadamente $25 \cdot 10^6 \cdot 230$ *bytes* (aproximadamente 5,52 *gigabytes*) de memória apenas para carregar um dos ativos em memória, antes de remover informações úteis e ordenar os dados.

Toda esta análise nos leva a uma simples conclusão: é extremamente custoso em termos computacionais ordenar grandes quantidades de dados, e, portanto, seria bem mais simples trabalhar com dados pré-processados, de forma que todo o trabalho "braçal" fosse realizado uma única vez. Neste sentido, os esforços que antes estavam voltados para a camada de dados do simulador passaram a ser investidos em uma estratégia bem mais simples, que envolveu criar uma ferramenta capaz de processar os dados necessários para uma simulação

e condensá-los em um único arquivo.

A forma mais simples de realizar esta tarefa sem consumir grandes quantidades de memória RAM consiste em dividir o conjunto de dados em pequenos arquivos, ordená-los individualmente e em seguida concatenar estes arquivos em um arquivo final ordenado. O diagrama da Figura 3.10 ilustra esse algoritmo.

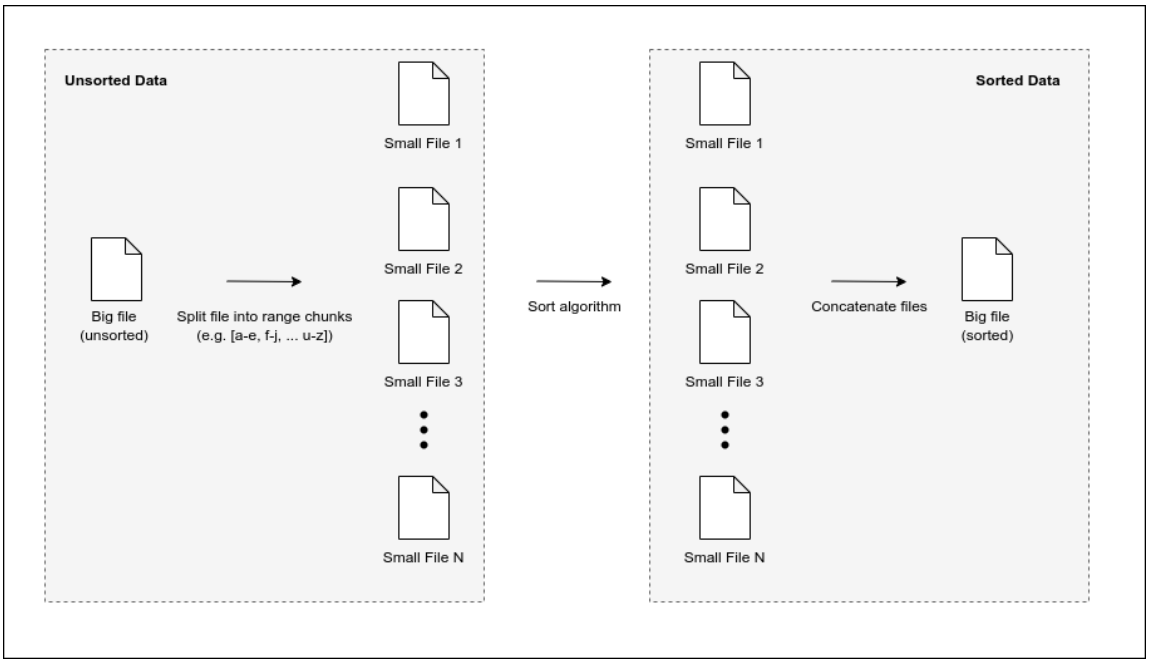

Figura 3.10 – Ordenação de dados

Fonte: Produzido pelo autor

Para o simulador, o objetivo principal é ordenar os dados com base no tempo prioridade das ordens de execução. Para isto, lemos todo o arquivo de dados uma única vez e guardamos cada linha em um arquivo específico. Na implementação realizada, cada arquivo armazena um conjunto de dados no intervalo de 2 horas, isto é, dados com um intervalo de tempo entre 8 e 10 horas (não incluso) são armazenados em um arquivo, dados entre as 10 e as 12 horas são registrados em outro arquivo e assim sucessivamente. Ao fim, temos um único arquivo ordenando contendo as ordens pertencentes apenas a um ativo predefinido. Desta forma, não há a necessidade do simulador ler (e ignorar) ordens que não são do interesse da simulação, e, portanto não há perda de tempo com processamento inútil.

Uma vez que todo pré-processamento é feito, o simulador está pronto para começar a sua execução. A rotina principal de leitura na camada de dados é extremamente simples, e consiste em um único *loop* de execução que lê linha a linha as ordens de execução, adiciona a ordem (sem processá-la) à fila de ordens e em seguida suspende sua execução por um curto intervalo de tempo, através da função *thread sleep*. O tempo em que a *thread* permanece suspensa é determinado por dois fatores: o intervalo de tempo real entre duas ofertas consecutivas, isto é, a diferença de tempo prioridade da oferta n e da oferta $n - 1$,

em millisegundos, e do parâmetro *simulation speed*, definido no arquivo de configuração. Este parâmetro funciona como um multiplicador do tempo de espera real entre duas leituras de dados consecutivas, podendo reduzir, igualar ou acelerar a velocidade da simulação em relação à execução real dos dados históricos. Vale ressaltar que apesar do tempo de leitura poder ser reduzido até zero, o grande gargalo de performance do simulador se encontra na etapa de processamento dos dados. A relação leitura *versus* processamento tende a ser bastante desequilibrada, o que limita a velocidade de execução das simulações. Mais detalhes relacionados à performance do simulador serão tratados na seção de resultados.

3.4.2.2 Camada de Aplicação

A camada de aplicação é onde encontra-se todo a lógica e as regras de negócio envolvidas na dinâmica de montagem do livro de ofertas e do fechamento de negociações. A rotina principal da camada de aplicação consiste em um *loop* de execução semelhante ao da camada de dados, consumindo os elementos da fila de ordens. Para cada iteração da execução, uma ordem é removida do início da fila e transformada do formato de *string* para um objeto com campos de tipos bem definidos. A partir deste objeto, inicia-se o processo de execução da ordem, que consiste em, primeiramente, identificar o tipo da ordem (neste caso, o tipo de execução: cancelamento, preenchimento, expiração, etc) e em seguida chamar a função responsável em lidar com o tipo de ordem.

A implementação desta camada baseia-se inteiramente no ciclo de vida de uma ordem, definido pela própria B3 em sua documentação pública. A Figura 3.11, disponível em B3 (2021) ilustra este ciclo de vida.

Do diagrama de transições da Figura 3.11, extraímos informações suficientes para desenvolver toda a lógica relacionada à dinâmica do livro. Os seguintes *order status* definem o fluxo de execução:

- *New*: novas ordens de compra e venda podem resultar em diferentes execuções do lado do servidor, a depender do estado da simulação. Ordens de compra com preço igual ou superior ao ASK irão gerar novas negociações, assim como, analogamente, ordens de venda com preço igual ou inferior ao BID também irão. Para estes casos e para o caso de ofertas à mercado, novas negociações são realizadas, e o preenchimento de ofertas será feito enquanto o preço de BID for maior ou igual ao preço de ASK, ou até todas as ofertas serem preenchidas completamente.

  Para todos os cenários diferentes dos citados acima, a ordem será adicionada ao livro de ofertas, e sua posição no livro será definida, primeiramente, pela atratividade do seu preço. Ofertas de compra são ordenadas no livro em ordem decrescente do valor de compra, enquanto ordens de venda são ordenadas em ordem crescente. O critério

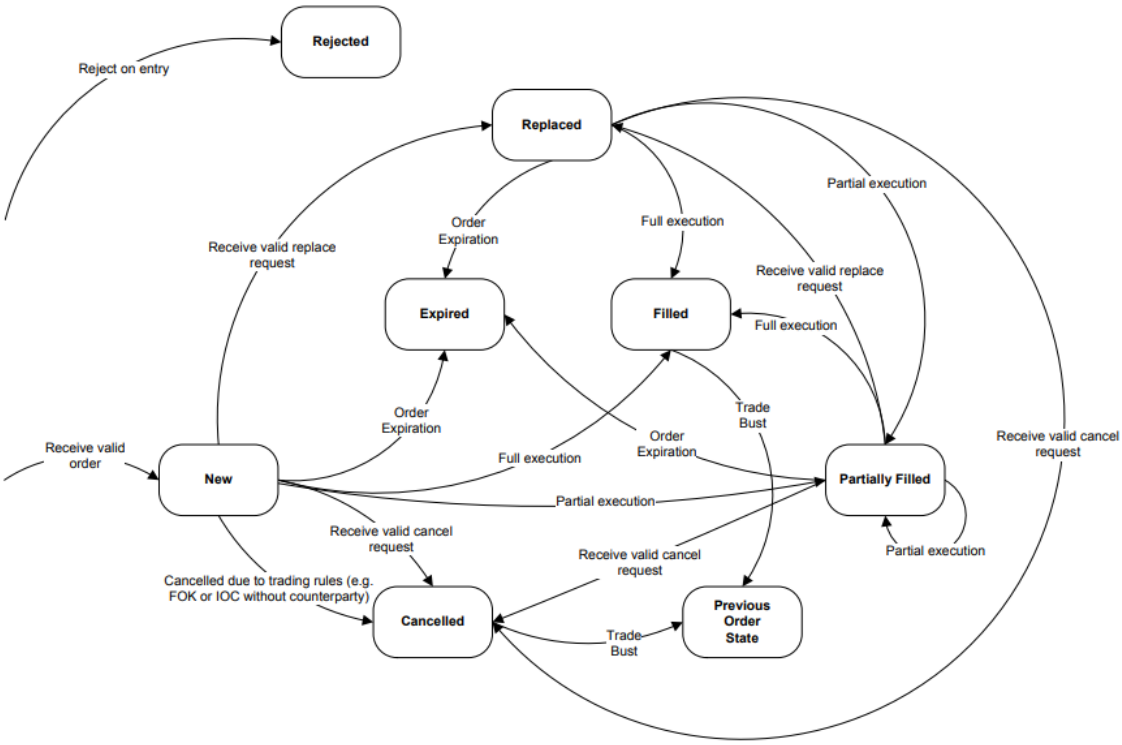

Figura 3.11 – Ciclo de vida de uma ordem

Fonte: B3 (2021, Figura 3)

de desempate é o tempo prioridade da ordem, definido pelo instante de tempo em que a oferta foi adicionada à fila.

- *Replaced*: ordens que já fazem parte do livro de ofertas e que ainda não foram executadas podem ser alteradas à qualquer momento. Conforme a especificação B3 (2021, seções 6.5 e 6.6), são permitidas diferentes mudanças nas ordens, e dependendo do tipo, estas mudanças podem resultar em uma perda de prioridade de execução. A Tabela 3.1 resume as alterações que possuem relevância para o funcionamento do simulador e o seu impacto na prioridade da ordem.

Tabela 3.1 – Alteração na prioridade de ordens

Tipo de alteração	**Influência sobre a prioridade**
Preço	Perda de prioridade
Aumento de quantidade	Perda de prioridade
Diminuição de quantidade	Mantém-se a prioridade
Tipo de ordem	Mantém-se a prioridade

Fonte: B3 (2021, seção 6.6)

- *Cancelled* ou *Expired*: o cancelamento de uma ordem ou sua expiração possuem o mesmo resultado prático na simulação: remoção da ordem do livro de ofertas. Um ponto que deve ser considerado é que, assim como no caso de alterações em ordens já existentes, é necessário primeiramente localizar esta ordem no livro de ofertas. Em

razão da forma com que os dados estão ordenados, esta operação pode ser relativamente custosa em termos de tempo computacional.

- *Rejected* ou *Filled*: as negociações de um ativo são realizadas automaticamente pelo simulador, e, portanto, os registros de preenchimento de ordens (parcial e completo) são simplesmente ignorados pela camada de aplicação. O mesmo serve para ordens rejeitadas pela B3, que nunca chegam a ser preenchidas ou a compor o livro de ofertas.

Com exceção das ordens ignoradas pelo sistema, todos os fluxos de execução alteram o livro de ofertas em algum nível, seja pela adição, preenchimento ou remoção de ordens. Como citado anteriormente, a ordenação das ofertas que compõem o livro é realizada segundo o preço do ativo, e, para cada ordem, um identificador único é atribuído. Por conta disso, operações que envolvam a busca de um elemento na fila de ordens a partir de seu identificador tendem a ser relativamente custosas, a depender da quantidade de ofertas no livro no momento da operação. No pior dos casos, operações de substituição, cancelamento ou expiração de ordens possuem uma complexidade computacional de $O(n)$, o que para valores suficientemente grandes de n pode influenciar significativamente na performance do simulador. Na seção de resultados serão apresentados *benchmarks* de simulação, onde o tamanho médio do livro durante a execução será um parâmetro considerado. Além disso, a lógica por trás da camada de aplicação consiste em uma simples laço de execução, o que faz com que todo trabalho de busca em estruturas de dados tenha que ser feito constantemente. Em contrapartida, para operações de inserção (mais frequentes nos arquivos de dados), o simulador comporta-se com uma performance excelente, dado que a estrutura de dados que representa esta fila de ofertas realiza uma busca binária nos preços, e, no pior dos casos, a complexidade de execução da adição de uma ordem ao livro é $O\ log(n)$.

Outro fator a ser levado em consideração consiste na metodologia de precificação das negociações. No ambiente real da bolsa de valores brasileira, utiliza-se um indicador de agressividade para determinar a oferta que possui prioridade em relação aos preços. Em termos simples, a oferta que está “agredindo” o livro possui prioridade, onde, para uma oferta de compra, o menor preço entre compra e venda será tomado, enquanto que para ofertas de venda, o maior preço será utilizado. Por se tratar de uma aproximação, onde fatores externos como latência e problemas de comunicação são praticamente inexistentes, a ordem com que as ofertas são preenchidas pelo simulador não é exatamente igual àquela realizada pela B3, e, portanto o indicador de agressividade da ordem não reflete, necessariamente, sua prioridade.

Empiricamente, observou-se que a melhor forma de aproximar os preços de negociação da simulação dos dados reais é tomando como ordem prioritária aquela cujo valor de compra/venda está mais próximo do último preço praticado. Dessa forma, observou-se uma maior proximidade dos resultados com os valores esperados.

3.4.2.3 Camada de Logging

Na camada de *logging* encontram-se os serviços de métricas e logs da aplicação. O serviço de métricas consiste em um laço de repetição com iterações executadas em espaços iguais de tempo, e é responsável por armazenar dados relacionados ao estado da aplicação em um arquivo de texto. Neste arquivo são registradas informações como tamanho da fila e do livro de ofertas, velocidade média de processamento de ofertas e o *throughput* da camada de dados, que funciona como um indicador da dinâmica de leitura e processamento de ordens. Já no serviço de logs estão implementadas funções que auxiliam o usuário a entender o estado corrente da simulação, mostrando informações como porcentagem total de ordens lidas, volume de mercado de um ativo, tamanho da fila de ofertas e do livro, etc. Estas informações são apresentadas para o usuário no terminal da aplicação, e servem como indicador do tempo total estimado de simulação.

3.4.2.4 Camada de Middleware

A camada de *middleware* implementa uma conexão via TCP *socket* que permite com que o servidor e o cliente troquem dados livremente. Uma vez que a conexão é estabelecida e o endereço do *socket* é definido, iniciam-se dois serviços, uma para recebimento de ordens (*ServerOrderReceiver*) e um para o envio de dados (*ServerResponseSender*). Este último é injetado em outras camadas da aplicação, e possui uma *callback* que permite o envio de dados no formato JSON para o cliente da aplicação.

3.4.3 Utilitários e Variáveis compartilhadas

Como o simulador foi implementando utilizando a linguagem de programação C++, fez-se o uso de classes e objetos para abstrair todas as funcionalidades comuns a mais de um serviço, com objetivo de facilitar a leitura e manutenibilidade do código. A seguir estão listados alguns dos principais módulos e estruturas do servidor.

- *StringUtils*, *DateUtils*, *ArrayUtils* e *OrderUtils*: classes com funcionalidades comuns à toda a aplicação, voltadas para a realização de operações não nativas da linguagem sobre estruturas de dados primitivas.
- *Clock*, *Config*, *Order* e *Context*: abstrações de estruturas necessárias para execução das simulações. Respectivamente, representando o relógio da aplicação, o arquivo de configurações, uma ordem de compra ou venda e o contexto da execução (quantidade processada, ativo alvo, etc).
- *Semaphore*, *OrdersQueue* e *Book*: estruturas de dados compartilhadas entre todas as camadas da aplicação, responsáveis por armazenar o estado do mercado em um determinado instante de tempo.

3.5 Cliente

A arquitetura do cliente da aplicação segue o mesmo padrão do servidor, porém com algumas ressalvas. Diferente do servidor, no cliente não há disputa por recursos, uma vez que, como explicado anteriormente, na estrutura "invertida" de cliente-servidor, o cliente comporta-se de maneira reativa, executando trechos de código conforme os dados recebidos do servidor. A Figura 3.12 ilustra esta arquitetura, destacando a existência de apenas duas *threads* concorrentes, uma para a criação do *socket* e outra para a execução de um servidor *web* com uma interface gráfica mínima.

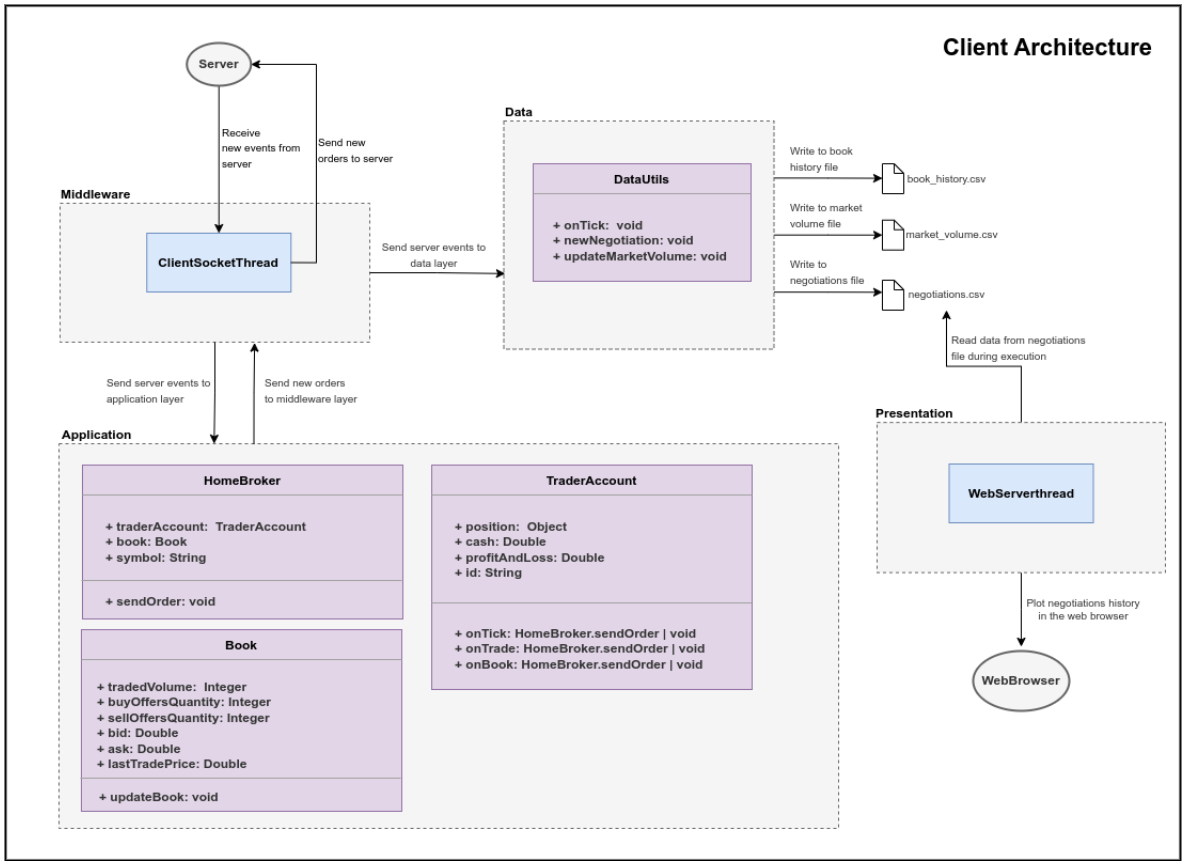

Figura 3.12 – Arquitetura do cliente

Fonte: Produzido pelo autor

A divisão de camadas do cliente é ligeiramente diferente do servidor, com uma camada de apresentação (*Presentation*) no lugar da camada de *Logging*. Assim como no servidor, é na camada de *Middleware* onde se encontra a lógica de comunicação externa, com a diferença de que esta camada comunica-se diretamente com as camadas de dados e aplicação, chamando funções nos componentes que as compõe à medida que novos dados são recebidos. A camada de dados tem papel diferente no cliente, já que ao invés de acessar e ler dados de arquivos ela é responsável por criar e popular arquivos com dados da execução. Em especial, quando uma simulação é iniciada e a conexão com o servidor estabelecida,

três arquivos são criados: *book history*, que armazena as oscilações nos valores de *bid* e *ask* ao longo do tempo, *market volume*, que registra o volume de preços de um determinado ativo em intervalos igualmente espaçados no tempo e por último o arquivo *negotiations*, que registra todas as negociações que ocorram ao longo da simulação e o seu tempo fictício de execução. A camada de apresentação fornece uma UI (*User Interface*) mínima para o cliente, contando com um servidor *web* para plotagem das negociações e uma janela interativa que permite ao usuário enviar ordens de compra e venda ao servidor.

Por fim, temos a camada de aplicação, que conta com um componente principal (*HomeBroker*) e dois componentes secundários, um representando o livro de ofertas simplificado e outro representando a conta do usuário, onde são armazenadas informações referentes ao balanço monetário, lucros e perdas e, principalmente, onde é implementada a estratégia de negociações do usuário.

3.5.1 Implementação

3.5.1.1 Middleware

Assim como no servidor, no cliente a camada de *middleware* também ocupa um papel central no transporte de dados. Ao receber um *payload* do servidor, o cliente inicia uma sequência de *callbacks*, que passa primeiro pela camada de aplicação e em seguida pela camada de dados. Novas ordens de compra são enviadas ao servidor a partir da execução da estratégia definida na camada de aplicação, ou manualmente, utilizando a interface gráfica definida pela camada de apresentação.

3.5.1.2 Data

A camada de dados lida com o armazenamento de informações em arquivos de texto, possibilitando um pós-processamento dos dados. Arquiteturalmente, optou-se por delegar a função de registro de dados para o cliente, eliminando um número considerável de operações de escrita sobre arquivos, que poderiam impactar negativamente na performance do simulador. Além da definição lógica, a camada de dados conta também com uma série de *scripts* para organização, análise e plotagem de dados de simulação. Os resultados gráficos apresentados na seção de resultados foram gerados através destes mesmos *scripts*.

3.5.1.3 Presentation

A camada de apresentação fornece uma interface amigável para o usuário, permitindo a visualização, em tempo real, da dinâmica de negociações, gerada a partir de um servidor *web* que renderiza o histórico de compras e vendas em uma janela do navegador. Além disso, a camada de apresentação também define uma ferramenta de envio de ordens para o

servidor, permitindo a realização de testes manuais de operações de compra e venda durante a simulação de estratégias.

A Figura 3.13 ilustra a interface gráfica do servidor *web*, apresentando a curva de preços de um ativo ao longo da execução da simulação.

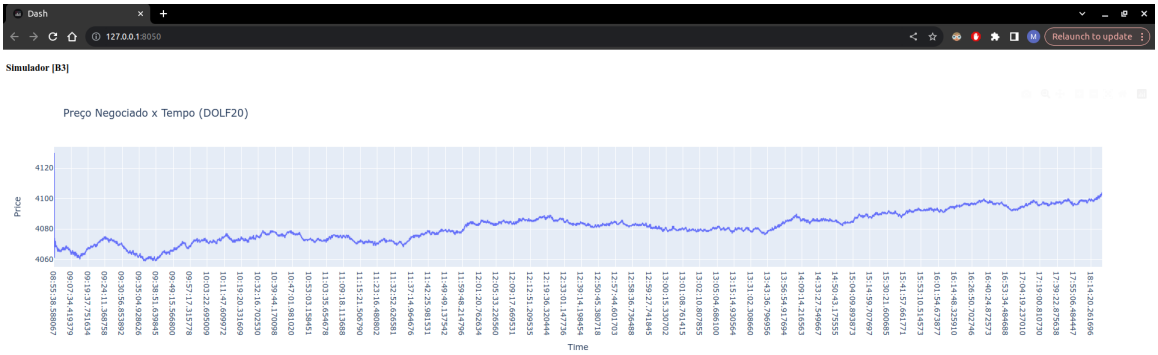

Figura 3.13 – Servidor *web* com histórico de negociações em tempo real

Fonte: Produzido pelo autor

Já a Figura 3.14 apresenta a ferramenta manual de envio de ordens do cliente, que permite a criação de ordens à mercado (enviadas com preço igual a zero) e ordens limite.

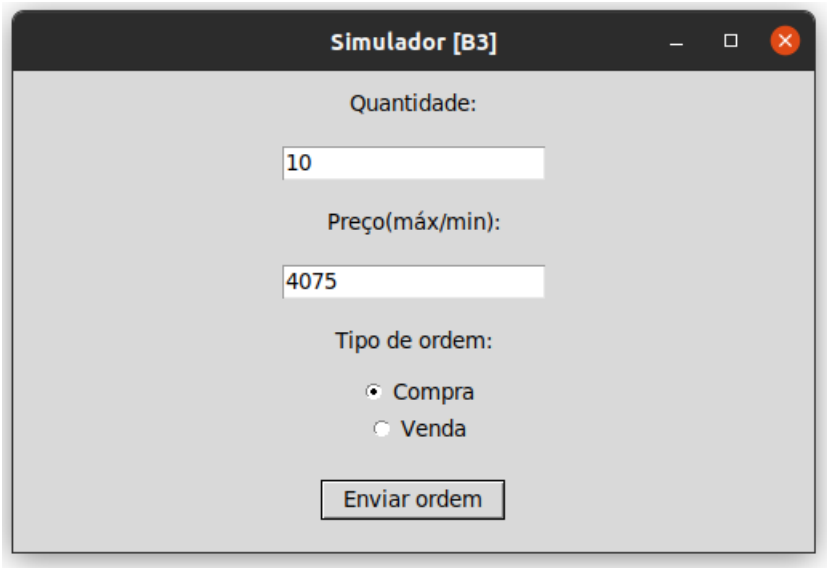

Figura 3.14 – Interface de envio de ordens

Fonte: Produzido pelo autor

3.5.1.4 Application

Na camada de aplicação encontra-se toda a lógica referente à execução da estratégia de *trading* algorítmico. Nela estão definidas as seguintes estruturas:

- *HomeBroker*: classe responsável por representar a estrutura de uma plataforma de *trading* (corretora), encapsulando uma abstração simplificada do livro de ofertas e gerenciando a conta do operador.
- *TraderAccount*: abstração da conta do operador em uma corretora de investimentos. Abriga propriedades básicas, relativas ao estado da conta fictícia, saldo, posição, perdas e ganhos, etc. Além disso, é nesta classe onde são definidas as *callbacks* que serão executadas quando a camada de *middleware* identificar o recebimento de eventos relevantes para a estratégia.
- *Book*: abstração do livro de ofertas simplificado, contendo apenas os preços de *ask* e *bid*, volume de *trade* e outras propriedades relevantes.

3.5.2 Interface de comunicação

Apesar da existência de protocolos específicos para esta finalidade, como o FAST, optou-se por simplificar o desenvolvimento das interfaces de comunicação a partir da estrutura de JSONs. No sentido cliente-servidor, o único dado enviado refere-se à criação de novas ordens. A estrutura do JSON consiste apenas em um campo do tipo *string*, formatado exatamente como nos arquivos de dados da B3. Já no sentido servidor-cliente, diferentes eventos com diferentes propriedades podem ser enviados. A Tabela 3.2 ilustra os possíveis tipos de eventos enviados pelo servidor ao cliente, além de uma explicação simples do que cada um destes eventos representa.

Tabela 3.2 – Interface de comunicação

Tipo de evento	**Definição**
UPDATE BOOK	Mudança no livro
NEW NEGOTIATION	Nova negociação
UPDATE MARKET VOLUME	Atualização do volume de ofertas
PURCHASE OFFER PARTIALLY FILLED	Compra parcialmente preenchida
PURCHASE OFFER ENTIRELY FILLED	Compra inteiramente preenchida
SALE OFFER PARTIALLY FILLED	Venda parcialmente preenchida
SALE OFFER ENTIRELY FILLED	Venda inteiramente preenchida
SIMULATION END	Fim da simulação

Fonte: Produzido pelo autor

Com exceção do evento *update market volume*, que é enviado ao cliente em intervalos iguais de tempo, todos os outros eventos dependem da dinâmica do mercado. Os eventos de preenchimento de ofertas são referentes apenas às ordens geradas pelo cliente, e não todas as ordens processadas pelo servidor.

Ordens enviadas pelo cliente ao servidor recebem uma *flag* que permite identificar sua origem. Independentemente do tipo da ordem, envia-se uma notificação do servidor para o cliente sempre que novas atualizações relacionadas à execução da oferta são observadas, informando o tipo da execução, o identificador da ordem e o momento da execução.

4 Resultados e análises

O Capítulo 4 apresenta uma análise dos resultados obtidos através de simulações realizadas sobre dois ativos: WING20 e DOLF20. A escolha destes dois ativos foi feita exclusivamente com base em sua alta liquidez, tendo em vista que ao classificar o conjunto de dados por quantidade de negociações realizadas obtém-se os dois ativos nas primeiras colocações.

4.1 Análise de dados

4.1.1 Resultados do simulador

Primeiramente, será realizada a análise da relação entre os preços de BID e ASK gerados pelo simulador ao ler as ordens diretamente dos arquivos de dados. Como estamos analisando ativos com alta liquidez, espera-se observar um *spread* relativamente pequeno, onde os preços de BID são ligeiramente inferiores aos preços de ASK. A Figura 4.15 ilustra esta dinâmica para o ativo DOLF20. Os preços mostrados representam a média dos valores observados no intervalo de tempo de um minuto.

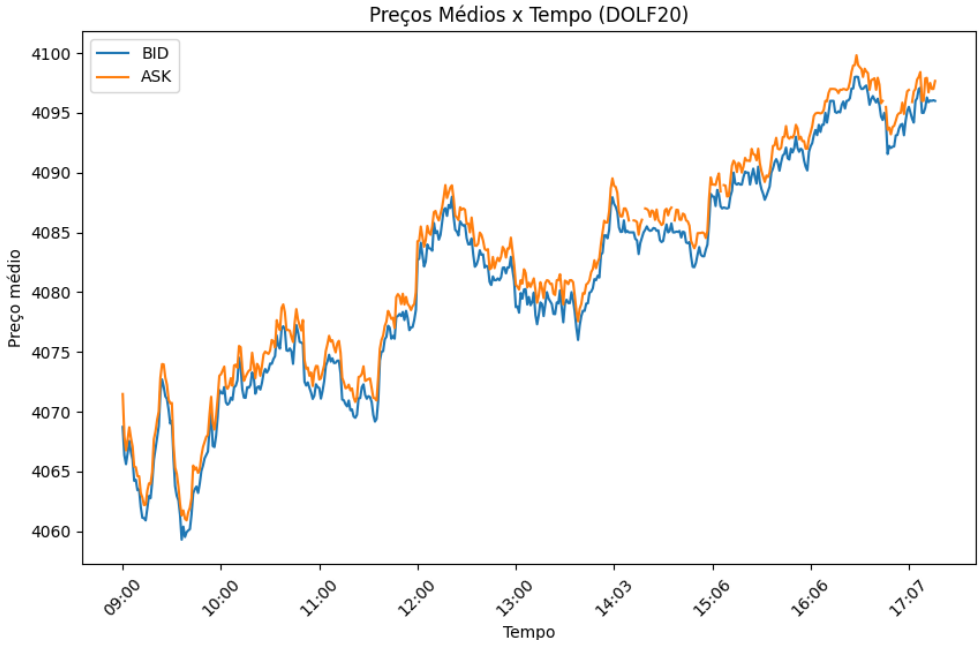

Figura 4.15 – Médias dos preços de ASK e BID (DOLF20)

Fonte: Produzido pelo autor

Como esperado, a diferença entre BID e ASK é relativamente pequena, e as curvas chegam a se sobrepor em alguns pontos. Na Figura 4.16 estão ilustrados os valores de BID e ASK para o ativo WING20, que possui uma quantidade de ordens aproximadamente 50 vezes maior que do ativo DOLF20. Para este conjunto de dados, a liquidez é tão grande que, para a escala de preços escolhida, onde a diferença entre os valores é de *R*$100,00, as curvas se sobrepõe.

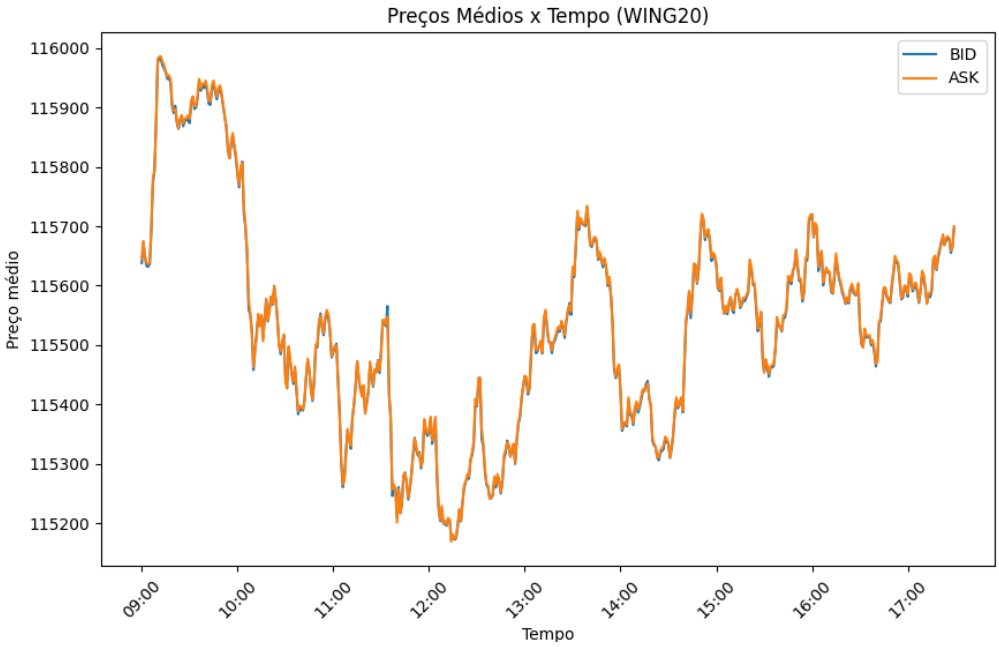

Figura 4.16 – Médias dos preços de ASK e BID (WING20)

Fonte: Produzido pelo autor

4.1.2 Comparação com dados históricos

A fim de validar a precisão do simulador, serão realizadas comparações de dados reais históricos e dados simulados, plotados em um mesmo gráfico.

Na Figura 4.17 estão plotados os preços de fechamento do ativo DOLF20, amostrados à cada minuto. A curva em azul representa os dados reais de negociação na data em questão, e a curva em laranja representa os registros de negociação gerados pelo simulador e armazenados em um arquivo pelo cliente.

Note-se que, apesar de haver diferenças de preços em alguns pontos, as curvas seguem o mesmo padrão, sem nenhum tipo de distorção no eixo do tempo. Em seguida, foi realizada a mesma análise para o ativo WING20, e os resultados podem ser observados na Figura 4.18.

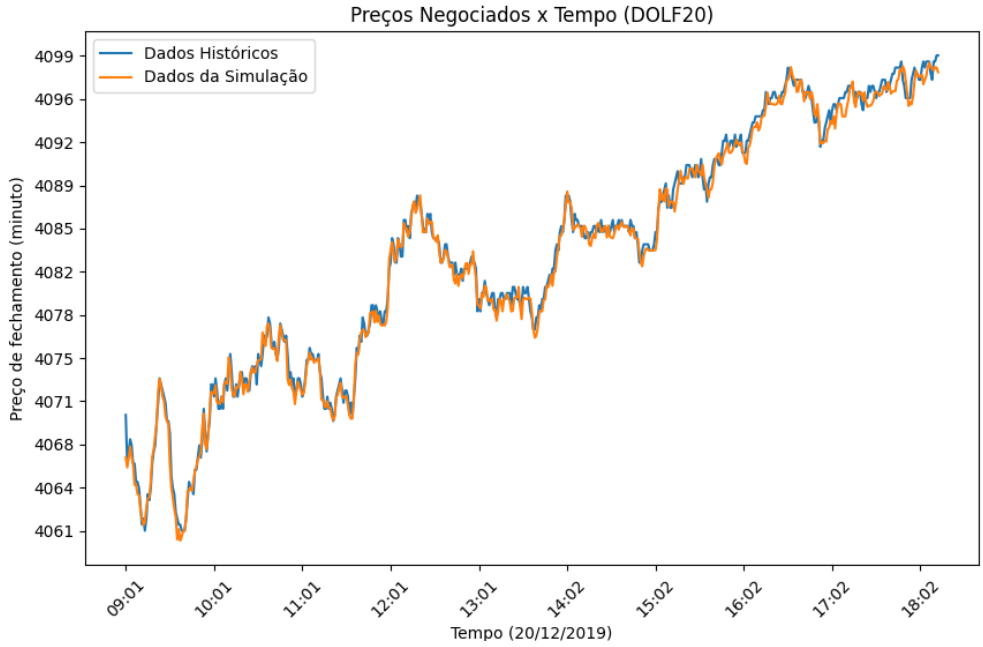

Figura 4.17 – Preços de fechamento (DOLF20)

Fonte: Produzido pelo autor

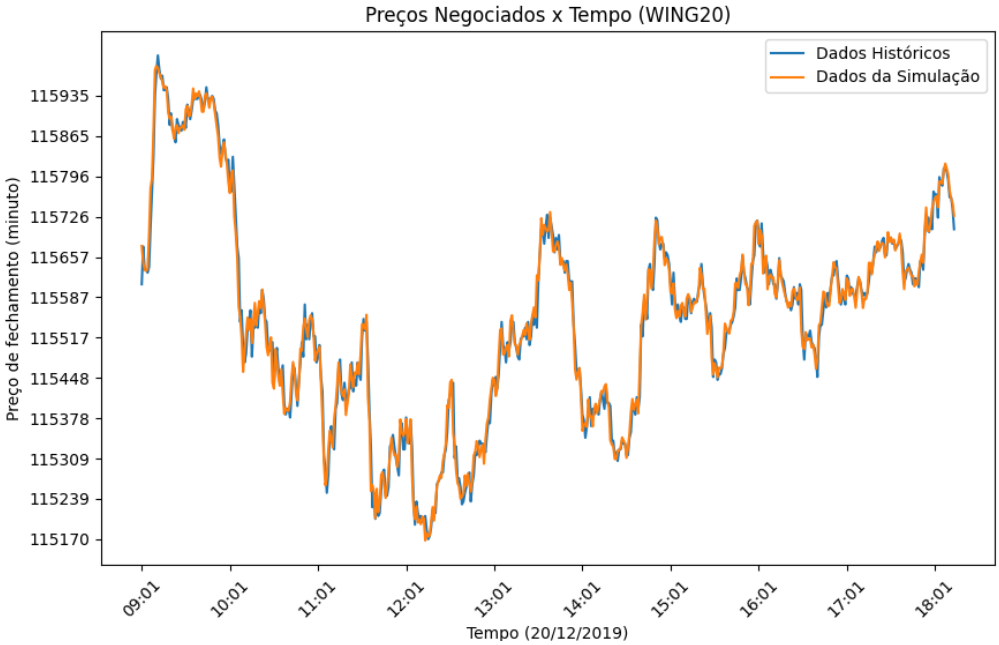

Figura 4.18 – Preços de fechamento (WING20)

Fonte: Produzido pelo autor

Ambos os resultados são bastante satisfatórios e mostram que, com algumas ressalvas, o modelo de preços utilizado e o mecanismo de sincronização de ofertas geram

resultados bastante próximos dos reais. Em um cenário ideal, as curvas seriam exatamente iguais, porém, por conta da diferença temporal na resolução dos dados utilizados (os dados simulados possuem uma resolução da ordem de millisegundos, enquanto que os dados reais representam os preços de fechamento à cada minuto) para construir os gráficos, verifica-se pequenas variações entre os valores reais e simulados, o que não gera impacto significativo na reprodução e teste das estratégias de *trading* algorítmico.

4.2 Insights e conclusões

No conjunto de arquivos gerados pelo cliente a partir de uma simulação, encontram-se os dados referentes as oscilações do mercado durante o período simulado. Em especial, estamos interessados em observar a dinâmica do livro de ofertas durante a execução, e para isso, plotamos estes dados no formato de mapas de calor. As Figuras 4.19 e 4.20 ilustram esta dinâmica, em que a intensidade dos *pixels* da imagem formada representam o volume de ofertas presentes no livro para um determinado preço e instante de tempo.

Como pode ser observado, a qualidade das imagens (densidade de *pixels*) é diretamente proporcional ao número de amostragens, e naturalmente, para ativos com maior liquidez, obtêm-se um melhor resultado. A interpretação destas imagens permite enxergar tendências de mercado, valores de suporte e também identificar posições fixas provavelmente provenientes de estudos de mercado, como nas linhas destacadas em um tom mais claro na Figura 4.20.

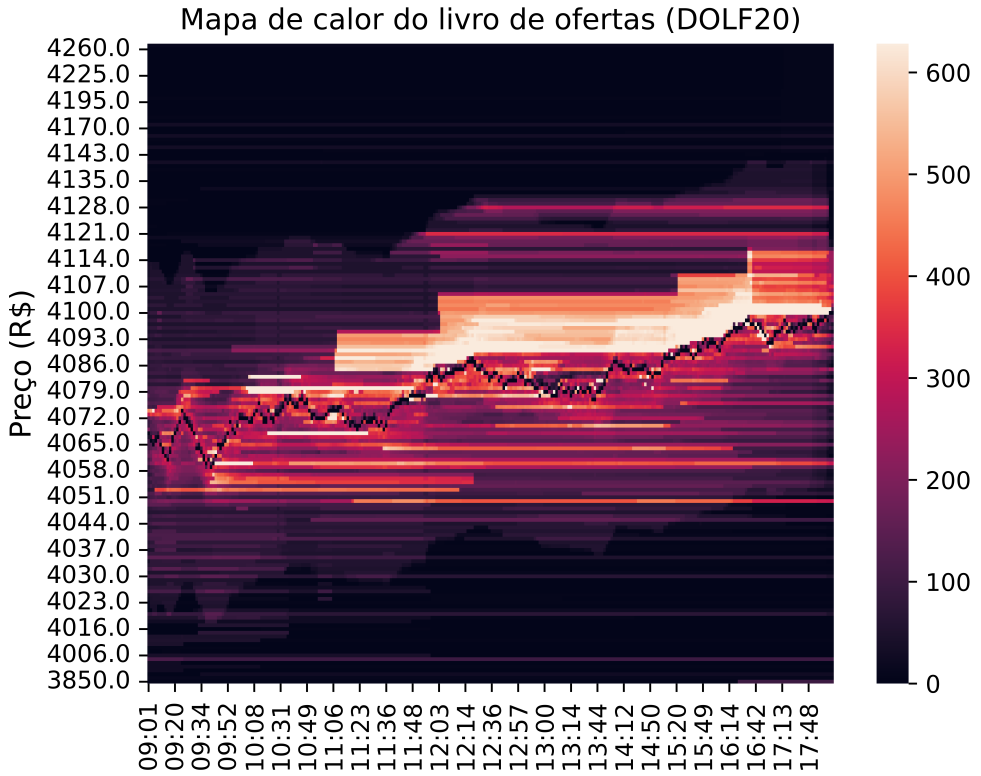

Figura 4.19 – Mapa de calor do livro de ofertas (DOLF20)

Fonte: Produzido pelo autor

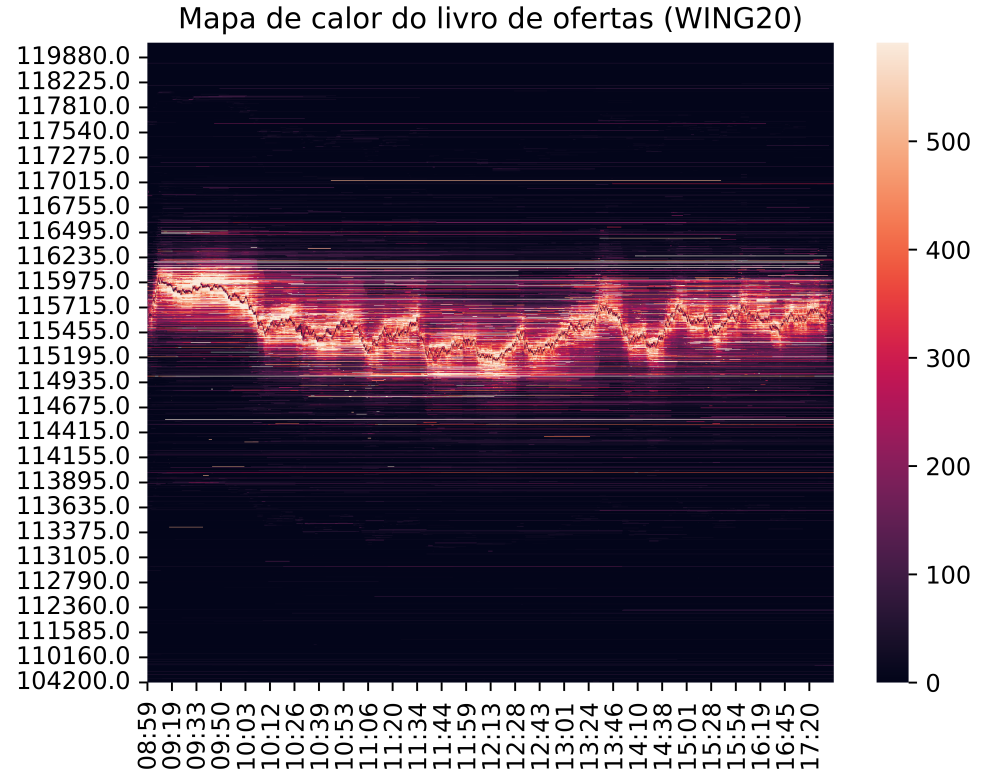

Figura 4.20 – Mapa de calor do livro de ofertas (WING20)

Fonte: Produzido pelo autor

4.3 Performance

Como citado nas seções anteriores, a performance de uma simulação está diretamente ligada ao tempo médio de processamento de ofertas, que, por sua vez, depende diretamente do tamanho do livro. Outro fator importantíssimo para esta análise está ligado à quantidade de memória utilizada pelo simulador durante sua execução, e este fator está diretamente associado ao tamanho das estruturas de dados que armazenam o estado da simulação em memória. Para um mesmo conjunto de dados, a relação entre o tamanho da fila de ordens não processadas e do livro de ofertas depende diretamente da taxa de leitura *versus* a taxa de processamento. Pensando diretamente no código, isso traduz-se na relação entre o *sleep time* da *thread* de leitura de dados e o da *thread* de processamento de ofertas. A partir dos gráficos das Figuras 4.21 e 4.22, podemos observar a diferença na dinâmica de leitura e processamento, onde duas simulações distintas foram executadas. Na primeira execução simulou-se o tempo médio de leitura de ofertas considerando que a taxa de processamento das ordens é instantânea (desabilitando a *thread* de processamento). A segunda execução utilizou do mesmo conjunto de dados, porém, desta vez com a *thread* de processamento de ofertas ativa.

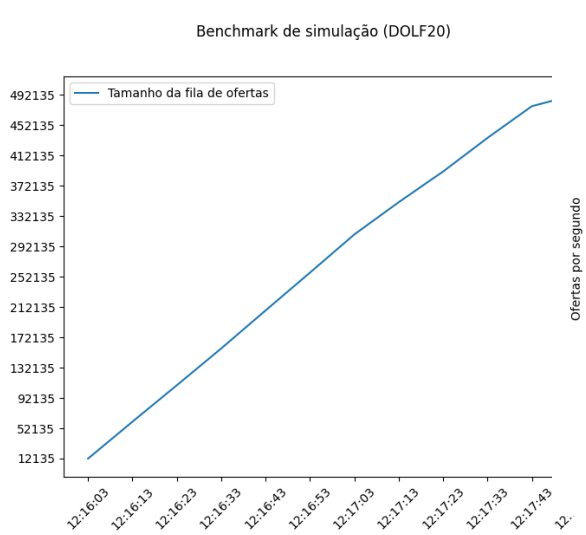

Figura 4.21 – Tamanho da fila de ordens

Fonte: Produzido pelo autor

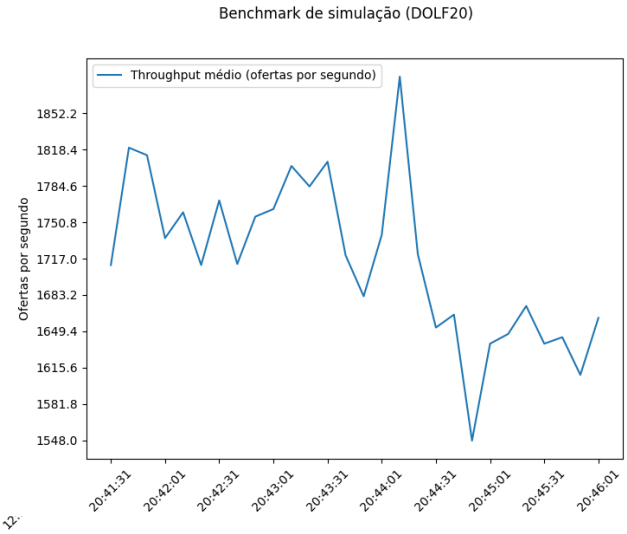

Figura 4.22 – Taxa de processamento da fila de ordens

Fonte: Produzido pelo autor

No gráfico da Figura 4.21 pode-se observar a velocidade praticamente constante de leitura, enquanto que no gráfico da Figura 4.22 observamos que a taxa de processamento de ofertas varia bastante, com um valor médio próximo à 1700 ofertas por segundo. Isto se dá devido ao fato de que a operação de leitura de ordens é sempre a mesma, consistindo apenas na adição de novas ordens ao fim da fila. Já na etapa de processamento, diferentes caminhos podem ser tomados, e consequentemente, o tempo computacional de cada execução varia. Com uma média de 1700 ofertas sendo processadas a cada segundo, podemos estimar que o

tempo médio de processamento de uma nova oferta gira em torno de 0.58ms, o que é um resultado satisfatório. Porém, vale ressaltar que esta velocidade de processamento depende diretamente do tamanho do livro, e que, quanto maior a liquidez do ativo, maior será o número de ofertas lidas, e consequentemente, maior será o tempo de processamento médio. Os gráficos da Figura 4.23 ilustram a relação entre a fila de ordens, o livro de ofertas e o tempo de processamento médio de novas ofertas.

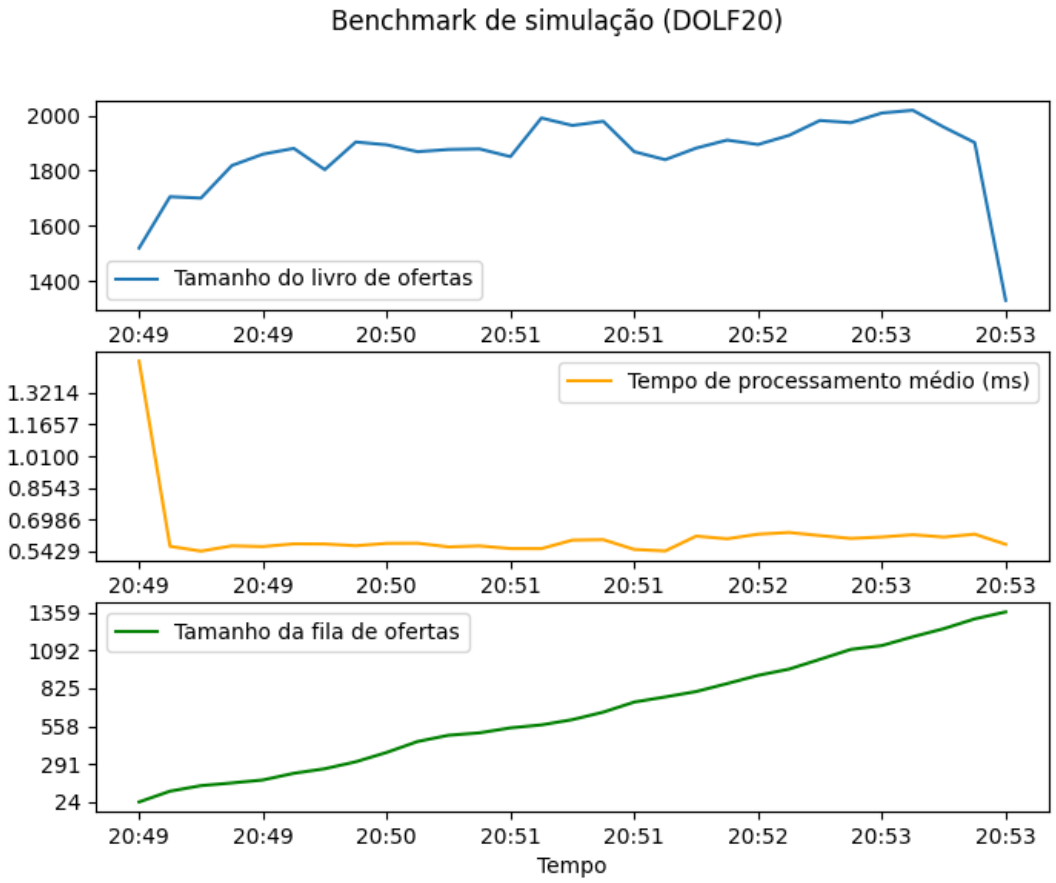

Figura 4.23 – *Benchmark* de simulação (DOLF20)

Fonte: Produzido pelo autor

Para esta simulação, o livro possui um tamanho médio de aproximadamente, 1863 ofertas, enquanto que o tempo médio de processamento é da ordem de 0,61ms, bem próximo do calculado com base no *throughput* médio de ofertas. Note que, o tamanho da fila de ordens aqui também cresce de forma praticamente constante, sem chegar a atingir o valor máximo de 10000 ordens, estipulado em código. Para fins comparativos, executou-se outra simulação para este mesmo conjunto de dados, ignorando o tempo fictício de espera entre duas ordens consecutivas na *thread* de leitura de dados. Os resultados estão ilustrados na Figura 4.24.

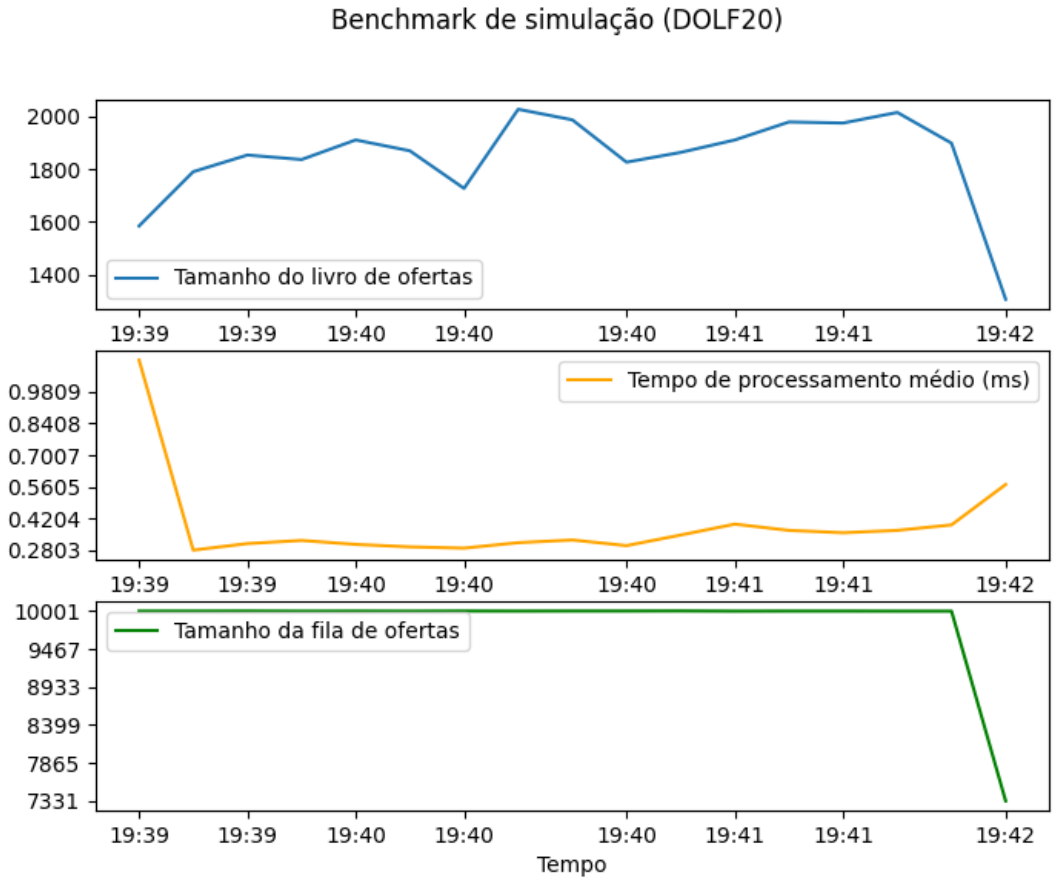

Figura 4.24 – *Benchmark* de simulação (DOLF20) - sem atraso entre leituras

Fonte: Produzido pelo autor

Nesta simulação, observa-se um tamanho do livro próximo aquele da simulação anterior, com um valor médio de 1845 ofertas. Em contrapartida, o tamanho da fila de ofertas atingiu seu valor máximo quase que instantaneamente, e o tempo médio de processamento caiu para 0,39*ms* (redução de 36%). Esta redução do tempo de processamento pôde ser observada porque na rotina de processamento de dados há uma condição em que, caso a fila de ofertas ultrapasse o valor máximo estipulado, executa-se um *loop* de processamento até que este valor seja respeitado, o que faz com que esta *thread* passe ainda mais tempo sendo executada, em relação à *thread* de leitura de dados. Apesar de apresentar uma melhora na performance do simulador, este resultado não pode ser considerado, já que, ao ignorar o tempo de espera entre ofertas consecutivas, gera-se inconsistências no gráfico de preços *versus* tempo do ativo, e além disso praticamente inviabiliza-se uma resposta em tempo hábil do lado do cliente.

Por fim, foi realizada a simulação de um ativo com uma liquidez consideravelmente superior ao anterior. Os gráficos da Figura 4.25 ilustram os resultados desta simulação.

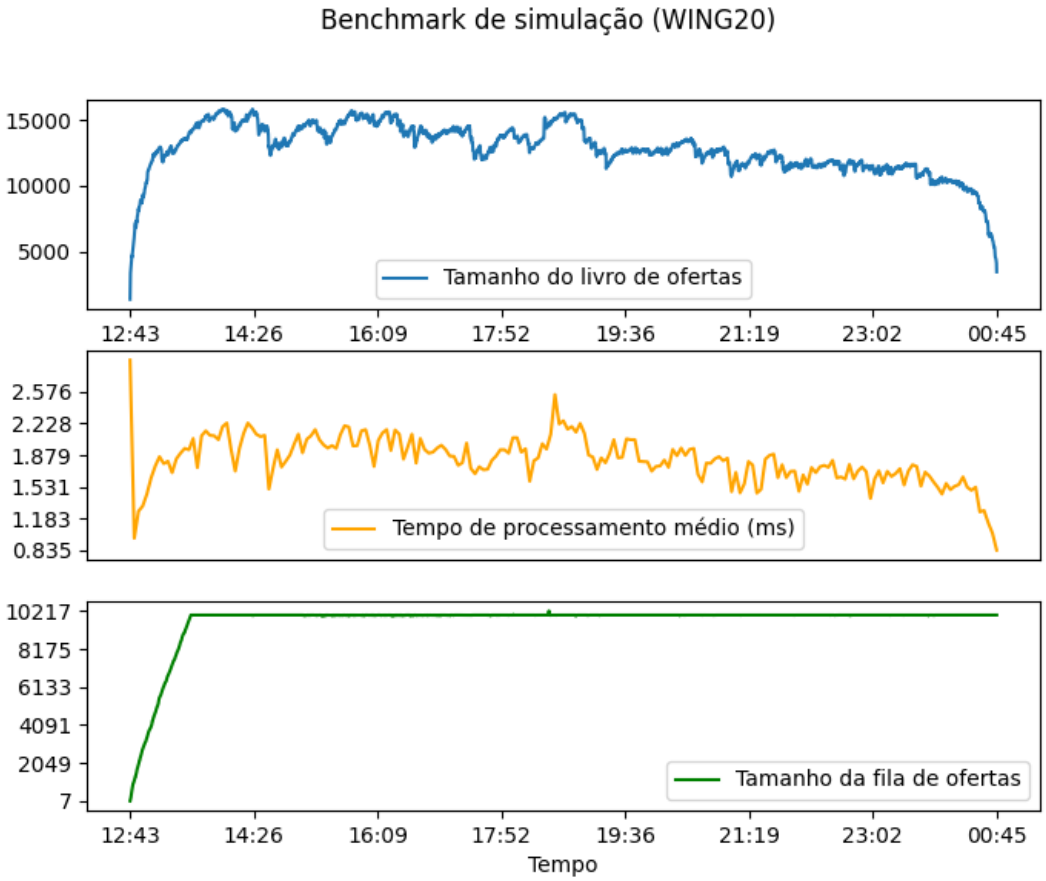

Figura 4.25 – *Benchmark* de simulação (WING20)

Fonte: Produzido pelo autor

Como pode-se observar, por conta do aumento do número de ofertas do livro, que agora possui um tamanho médio de aproximadamente 12800 ofertas, observou-se uma piora significativa do tempo de processamento, desta vez com uma média de aproximadamente 1,8ms. O tamanho da fila de ordens cresceu gradativamente durante o início da simulação, e estabilizou-se no valor máximo permitido ao longo do restante da execução, corroborando a hipótese inicial de que o gargalo de performance encontra-se no tamanho do livro de ofertas.

4.4 Avaliação do simulador

O conjunto de funcionalidades definido na seção 3.1.1 foi implementando por completo. Nas seções anteriores, foram apresentados gráficos gerados a partir de dados de simulação, e a posterior comparação com dados reais demonstra que as funções primárias do servidor foram implementadas com êxito.

Através dos resultados obtidos, e das análises realizadas, pode-se dizer que o simulador desenvolvido cumpre os objetivos definidos no escopo do projeto. Como o objetivo do

projeto era desenvolver um ambiente de negociações próximo ao da B3, não foram imprimidos grandes esforços na execução de testes de estratégias de *trading* algorítmico. Entretanto, o ambiente desenvolvido é capaz de lidar com ordens geradas a partir do cliente da mesma forma com que executa as ordens advindas dos arquivos de dados históricos. Sendo assim, propõe-se utilizar este simulador como arcabouço de testes em futuros trabalhos que explorem o tema de HFT, a fim de validar os resultados aqui obtidos.

Ainda assim, o simulador possui algumas limitações de performance e arquitetura a serem trabalhadas. A próxima seção resume alguns dos pontos que podem ser explorados em trabalhos futuros.

4.5 Melhorias futuras

Como recomendação para implementações futuras, sugere-se desenvolver e executar o seguinte conjunto de melhorias, listadas por prioridade:

1. Testes automatizados: desenvolver cenários de testes automatizados com o objetivo final de identificar e corrigir eventuais *bugs* que tenham sido adicionados ao código durante a etapa de desenvolvimento.

2. Arquitetura: revisitar a arquitetura do projeto, buscando aperfeiçoar as técnicas de engenharia de *software* utilizadas, delimitando de maneira clara a relação entre as camadas das aplicações, suas responsabilidades e mecanismos de comunicação.

3. Performance: explorar diferentes estruturas de dados, visando melhorar as operações mais custosas sobre o livro de ofertas. Além disso, redefinir algumas das interfaces e explorar boas práticas com a linguagem podem reduzir consideravelmente o tempo de processamento médio das ofertas. O objetivo principal é atingir um patamar onde diferentes estratégias possam ser testadas em curtos intervalos de tempo, tornando mais simples o processo de aperfeiçoamento.

4. Múltiplos ativos alvo: como citado anteriormente, a implementação atual do *software* não permite operar dois ativos ao mesmo tempo. Esta limitação está intimamente relacionada à forma como os dados são organizados antes de serem processados. A lógica de execução das camadas da aplicação está pautada em um único símbolo, portanto mudanças no código são necessárias.

5. Ordenação automática dos dados: antes de iniciar uma simulação, é necessário separar e organizar os dados que serão processados. Atualmente este processo é realizado manualmente, através da execução de um *script* em *Python*, mas idealmente espera-se que o programa seja capaz de realizar esta tarefa em uma etapa de pré execução.

6. Interface gráfica: o simulador conta com um servidor *web* local onde são plotados os dados de negociação em tempo real e uma interface simples onde é possível enviar ordens de compra e venda. Idealmente, estas interfaces deveriam estar centralizadas em um único módulo, onde outras informações relevantes como o topo do livro de ofertas, volume de *trading* e negociações também seriam apresentadas.

7. Definição interativa de estratégias: todo o código por trás das estratégias de *trading* é escrito manualmente e adicionado a uma classe do cliente antes de realizar uma simulação. Em um cenário ideal, esta definição poderia ser feita de forma interativa, via interface gráfica (por exemplo) a fim de viabilizar o uso do simulador para indivíduos que não possuem um *background* técnico em programação.

5 Conclusões

O software desenvolvido tem como principal objetivo fornecer um ambiente de simulação para estratégias de *trading* algoritmíco incluindo *trading* de alta frequência. Através dos módulos desenvolvidos, é possível reproduzir a dinâmica de mercado executada em um determinado momento da história a partir de um conjunto de dados do mundo real. A partir da execução do *replay* de mercado é possível testar e validar modelos de negociação projetadas para um determinado comportamento de mercado, e a partir dos resultados obtidos aprimorar e otimizar a estratégia antes de aplicá-la no mundo real.

Os resultados obtidos através da análise de dados provenientes de simulação são bastante próximos dos dados do mundo real, e nos permitem concluir que o *software* é capaz de reproduzir de maneira satisfatória a dinâmica do livro de ofertas e as negociações realizadas no ambiente real. As limitações de performance observadas estão intimamente relacionadas ao volume de dados processados, o que pode elevar consideravelmente o tempo total de simulação. Entretanto, para a proposta inicial que visa o *trading* algorítmico, o *software* se sai relativamente bem, permitindo a execução de centenas de milhares de ofertas no intervalo de minutos, tornando-o bastante útil no contexto de *trading* de alta frequência.

O trabalho realizado foi motivado principalmente pela escassez de simuladores financeiros disponíveis para o público geral capazes de lidar com as nuances de estratégias de negociação automatizadas. Neste sentido, pode-se considerar o desenvolvimento do simulador como a primeira etapa de um trabalho mais amplo, visando o desenvolvimento de ferramentas de *software* que possibilitem o estudo de modelos de finanças quantitativas implementadas através do emprego de técnicas de engenharia de *software*.

Na literatura relativa ao tema é possível encontrar uma diversa gama de estudos relacionados ao conceito de finanças quantitativas. Através do emprego de técnicas de inteligência artificial no desenvolvimento de estratégias de mercado, diversos modelos foram desenvolvidos ao longo das últimas décadas. Em (Nelson; Pereira; Oliveira, 2017), utiliza-se uma rede neural baseada em LSTM (*Long Short-Term Memory*) para prever preços futuros de ações utilizando como base dados históricos da B3 em conjunto com indicadores de análise técnica. Em (Li *et al.*, 2021) também utiliza-se uma rede LSTM para a predição de preços, e o treinamento da rede é feito com base em notícias *online* recentes envolvendo o mercado de ações.

Dentro do conceito de finanças quantitativas, outra área que vem se desenvolvendo ao longo dos últimos anos é o estudo da aplicação de conceitos de teoria de controle em modelos de *trading* algorítmico. Alguns dos exemplos de trabalhos que exploram esta área incluem (Barmish *et al.*, 2013), onde propõe-se uma arquitetura de simulação e análise de

performance baseada em *feedback-control* (Figura 5.26).

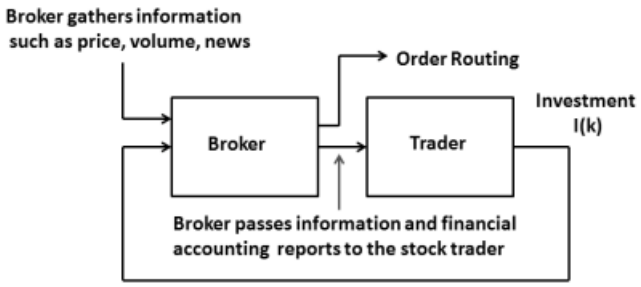

Figura 5.26 – Modelo de *feedback-control*

Fonte: (Barmish *et al.*, 2013)

Neste modelo, a cotação dos ativos e outras informações de mercado são dados externos ao sistema, e são constantemente repassadas à conta do *trader* pelo *broker*. O *trader* determina o nível de investimento a ser realizado através dos *inputs* fornecidos pelo *broker*, e repassa essa informação para o *broker*, fechando a malha de controle. Outros exemplos de estudos neste contexto podem ser encontrados na literatura, como em (Barmish, 2008) onde utiliza-se de técnicas de controle robusto para definir a estratégia de controle do sistema (mercado), e em (Malekpour; Primbs; Barmish, 2013) onde a ação de controle do modelo proposto em (Barmish *et al.*, 2013) é baseada em um controlador proporcional integral.

Dessa forma, vale salientar que a realização de trabalhos futuros que complementem este projeto não se restringe ao aprimoramento do simulador desenvolvido, e pode contemplar o emprego de diferentes modelos de investimento para comparação de resultados e otimizações de performance. Portanto, sugere-se a utilização deste software como uma ferramenta de apoio no desenvolvimento e análise de estratégias de investimento baseadas em *trading* algorítmico.

Referências

ALDRIDGE, I. **High-frequency Trading: A Practical Guide to Algorithmic Strategies and Trading Systems**. John Wiley & Sons, Incorporated, 2013. (Wiley trading series). ISBN 9781119203803. Disponível em: https://books.google.com.br/books?id=kHPwjgEACAAJ. Citado nas pp. 22 e 25.

B3. **A descoberta da bolsa pelo investidor brasileiro**. [*S.l.*], 2020. Disponível em: https://www.b3.com.br/data/files/E6/75/BC/90/36ECA71068C61CA7AC094EA8/Estudo_PF-final%20_mai.20_.pdf. Citado na p. 10.

B3. **EntryPoint: Order Entry Messaging**. [*S.l.*], 2021. Disponível em: https://www.b3.com.br/data/files/19/A3/1C/16/B721B71027085EA7AC094EA8/EntryPointMessagingGuidelines2.9.16.pdf. Citado nas pp. 37 e 38.

B3. **Derivativos - Conceitos**. [*S.l.*], 2023. Disponível em: https://www.b3.com.br/lumis/portal/file/fileDownload.jsp?fileId=8AE490CA6F165E34016F1E97CE4A2C63#:~:text=Contratos%20Derivativos%20representam%20ativos%20cujas%20cota%C3%A7%C3%B5es%20e%20pre%C3%A7os%20dependem%20(derivam,%2C%20%C3%ADndices%2C%20commodities... Citado na p. 14.

B3. **FIX/FAST Market Data Messaging Specification**. [*S.l.*], 2023. Disponível em: https://www.b3.com.br/data/files/90/C1/D8/C4/81C95810F534EB48AC094EA8/UMDF_MarketDataSpecification_v2.1.8%20draft.pdf. Citado na p. 25.

B3. **Perfil pessoas físicas**. [*S.l.*], 2023. Disponível em: https://www.b3.com.br/pt_br/market-data-e-indices/servicos-de-dados/market-data/consultas/mercado-a-vista/perfil-pessoas-fisicas/perfil-pessoa-fisica/. Citado na p. 10.

BARMISH, B. R. On trading of equities: A robust control paradigm. **IFAC Proceedings Volumes**, v. 41, n. 2, p. 1621–1626, 2008. ISSN 1474-6670. 17th IFAC World Congress. Disponível em: https://www.sciencedirect.com/science/article/pii/S1474667016391844. Citado na p. 57.

BARMISH, B. R.; PRIMBS, J. A.; MALEKPOUR, S.; WARNICK, S. On the basics for simulation of feedback-based stock trading strategies: An invited tutorial session. *In*: **52nd IEEE Conference on Decision and Control**. [*S.l.: s.n.*], 2013. p. 7181–7186. Citado nas pp. 56 e 57.

BRITANNICA. **Publican | Ancient Rome, Tax Collectors, Finances - Britannica**. [*S.l.*], 2023. Disponível em: https://www.britannica.com/topic/publican. Citado na p. 13.

CARTEA, Á.; JAIMUNGAL, S.; PENALVA, J. **Algorithmic and high-frequency trading**. [*S.l.*]: Cambridge University Press, 2015. Citado nas pp. 14, 15, 19 e 20.

CHEN, J. **Algorithmic Trading: Definition, How It Works, Pros & Cons**. [*S.l.*], 2022. Disponível em: https://www.investopedia.com/terms/a/algorithmictrading.asp. Citado na p. 21.

CHEN, J. **Backtesting: Definition, How It Works, and Downsides**. [*S.l.*], 2023. Disponível em: https://www.investopedia.com/terms/b/backtesting.asp#:~:text=Backtesting%20is%20the%20general%20method,to%20employ%20it%20going%20forward. Citado na p. 30.

COMMUNITY, F. T. **FAST Protocol**. [*S.l.*], 2016. Disponível em: https://www.fixtrading.org/standards/fast/. Citado na p. 25.

EISENHARDT, P. **Introduction to Clearing and Settlement**. [*S.l.*], 2023. Disponível em: https://financeunlocked.com/videos/securities-clearing-and-settlement. Citado na p. 17.

GOMBER, P.; ARNDT, B.; LUTAT, M.; UHLE, T. High-frequency trading. **SSRN Electronic Journal**, 01 2011. Citado na p. 22.

GORHAM, M.; SINGH, N. **Electronic Exchanges: The Global Transformation from Pits to Bits**. Elsevier Science, 2009. (Elsevier and IIT Stuart Center for Financial Markets Press). ISBN 9780080921402. Disponível em: https://books.google.com.br/books?id=vpTpY-HHWbkC. Citado na p. 16.

HISTORIA do Mercado de Capitais. [*S.l.*], 2023. Disponível em: https://www.gov.br/investidor/pt-br/investir/como-investir/conheca-o-mercado-de-capitais/historia-do-mercado-de-capitais#:~:text=No%20estado%20de%20S%C3%A3o%20Paulo,resultando%20no%20fechamento%20da%20Bolsa. Citado na p. 14.

INSTITUCIONAL (B3). [*S.l.*], 2023. Disponível em: https://www.b3.com.br/pt_br/b3/institucional/quem-somos/. Citado na p. 14.

LI, Q.; TAN, J.; WANG, J.; CHEN, H. A multimodal event-driven lstm model for stock prediction using online news. **IEEE Transactions on Knowledge and Data Engineering**, v. 33, n. 10, p. 3323–3337, 2021. Citado na p. 56.

MALEKPOUR, S.; PRIMBS, J. A.; BARMISH, B. R. On stock trading using a pi controller in an idealized market: The robust positive expectation property. *In*: **52nd IEEE Conference on Decision and Control**. [*S.l.: s.n.*], 2013. p. 1210–1216. Citado na p. 57.

NELSON, D. M. Q.; PEREIRA, A. C. M.; OLIVEIRA, R. A. de. Stock market's price movement prediction with lstm neural networks. *In*: **2017 International Joint Conference on Neural Networks (IJCNN)**. [*S.l.: s.n.*], 2017. p. 1419–1426. Citado na p. 56.

NUTI, G.; MIRGHAEMI, M.; TRELEAVEN, P.; YINGSAEREE, C. Algorithmic trading. **Computer**, v. 44, n. 11, p. 61–69, 2011. Citado na p. 22.

OVERFLOW, S. **Stack Overflow Developer Survey 2022**. [*S.l.*], 2022. Disponível em: https://survey.stackoverflow.co/2022#technology-most-popular-technologies. Citado na p. 29.

SCOTT, G. **Matching Orders: What They Are, How They Work, and Examples**. [*S.l.*], 2023. Disponível em: https://www.investopedia.com/terms/m/matchingorders.asp. Citado na p. 18.

SCOTT, G. **Open Outcry: What it is, How it Works, Decline in Popularity**. [*S.l.*], 2023. Disponível em: https://www.investopedia.com/terms/o/openoutcry.asp. Citado na p. 13.

SMITH, B. M. **A history of the global stock market: from ancient Rome to Silicon Valley**. [*S.l.*]: University of Chicago press, 2004. 9-11 p. Citado na p. 13.

TECHOPEDIA. **Co-location**. [*S.l.*], 2017. Disponível em: https://www.techopedia.com/definition/2485/co-location-colo. Citado na p. 23.

Apêndices

Apêndice A – Trechos de código

A.1 Estratégia de trading

Código A.1 – Conta do trader (simplificada)

```
import random
import json

TRADE_EVENTS = ["SALE_OFFER_ENTIRELY_FILLED",
   "SALE_OFFER_PARTIALLY_FILLED",
   "PURCHASE_OFFER_ENTIRELY_FILLED",
   "PURCHASE_OFFER_PARTIALLY_FILLED"]

class TraderAccount:
    def __init__(self, book, initial_cash = 1000000):
        self.INITIAL_CASH = initial_cash
        self.position = {}
        self.cash = self.INITIAL_CASH
        self.profitAndLoss = 0
        self.id = ''# random number
        self.book = book

    def handle_event(self, data):
        event = data["event"]
        if event in TRADE_EVENTS:
            self.onTrade(data)
        elif "UPDATE_BOOK":
            self.onBook(data)
        elif "ON_TICK":
            self.ontTick(data)

    def ontTick(self, data):
        # execute ontTick strategy
        return

    def onBook(self, data):
        # execute onBook strategy
        return

    def onTrade(self, data):
        # execute onTrade strategy
        return
```

A.2 Eventos gerados pelo servidor

Código A.2 – Exemplo de histórico de negociações (estratégia)

```
1  {'event': 'PURCHASE_OFFER_PARTIALLY_FILLED', 'id': '5227574675',
       'price': 4063.0, 'qty': 15, 'symbol': 'DOLF20', 'time':
       '09:35:12.664545'}
2  {'event': 'PURCHASE_OFFER_PARTIALLY_FILLED', 'id': '5227574675',
       'price': 4063.0, 'qty': 5, 'symbol': 'DOLF20', 'time':
       '09:35:12.664599'}
3  {'event': 'PURCHASE_OFFER_PARTIALLY_FILLED', 'id': '5227574675',
       'price': 4063.0, 'qty': 10, 'symbol': 'DOLF20', 'time':
       '09:35:12.664651'}
4  {'event': 'PURCHASE_OFFER_PARTIALLY_FILLED', 'id': '5227574675',
       'price': 4063.0, 'qty': 5, 'symbol': 'DOLF20', 'time':
       '09:35:12.664686'}
5  {'event': 'PURCHASE_OFFER_PARTIALLY_FILLED', 'id': '5227574675',
       'price': 4063.0, 'qty': 5, 'symbol': 'DOLF20', 'time':
       '09:35:12.664721'}
6  {'event': 'PURCHASE_OFFER_PARTIALLY_FILLED', 'id': '5227574675',
       'price': 4063.0, 'qty': 5, 'symbol': 'DOLF20', 'time':
       '09:35:12.664761'}
7  {'event': 'PURCHASE_OFFER_PARTIALLY_FILLED', 'id': '5227574675',
       'price': 4063.0, 'qty': 5, 'symbol': 'DOLF20', 'time':
       '09:35:12.664817'}
8  {'event': 'PURCHASE_OFFER_PARTIALLY_FILLED', 'id': '5227574675',
       'price': 4063.0, 'qty': 5, 'symbol': 'DOLF20', 'time':
       '09:35:12.664879'}
9  {'event': 'PURCHASE_OFFER_PARTIALLY_FILLED', 'id': '5227574675',
       'price': 4064.0, 'qty': 5, 'symbol': 'DOLF20', 'time':
       '09:35:12.664939'}
10 {'event': 'PURCHASE_OFFER_PARTIALLY_FILLED', 'id': '5227574675',
       'price': 4064.0, 'qty': 5, 'symbol': 'DOLF20', 'time':
       '09:35:12.664993'}
11 {'event': 'PURCHASE_OFFER_PARTIALLY_FILLED', 'id': '5227574675',
       'price': 4064.0, 'qty': 5, 'symbol': 'DOLF20', 'time':
       '09:35:12.665049'}
12 {'event': 'PURCHASE_OFFER_PARTIALLY_FILLED', 'id': '5227574675',
       'price': 4064.0, 'qty': 10, 'symbol': 'DOLF20', 'time':
       '09:35:12.665100'}
13 {'event': 'PURCHASE_OFFER_PARTIALLY_FILLED', 'id': '5227574675',
       'price': 4064.0, 'qty': 10, 'symbol': 'DOLF20', 'time':
       '09:35:12.665139'}
14 {'event': 'PURCHASE_OFFER_PARTIALLY_FILLED', 'id': '5227574675',
       'price': 4064.0, 'qty': 5, 'symbol': 'DOLF20', 'time':
       '09:35:12.665174'}
15 {'event': 'PURCHASE_OFFER_ENTIRELY_FILLED', 'id': '5227574675',
       'price': 4064.0, 'qty': 5, 'symbol': 'DOLF20', 'time':
       '09:35:12.665208'}
16 {"id": "0400401321", "cash": 593655.0, "profitAndLoss": 0.999955,
       "position": {"DOLF20": 100}}
17 {'event': 'SALE_OFFER_PARTIALLY_FILLED', 'id': '7545615515',
       'price': 4097.0, 'qty': 5, 'symbol': 'DOLF20', 'time':
       '16:42:04.912111'}
```

```
18 {'event': 'SALE_OFFER_PARTIALLY_FILLED', 'id': '7545615515',
       'price': 4097.0, 'qty': 50, 'symbol': 'DOLF20', 'time':
       '16:42:04.912199'}
19 {'event': 'SALE_OFFER_PARTIALLY_FILLED', 'id': '7545615515',
       'price': 4097.0, 'qty': 5, 'symbol': 'DOLF20', 'time':
       '16:42:04.912280'}
20 {'event': 'SALE_OFFER_PARTIALLY_FILLED', 'id': '7545615515',
       'price': 4097.0, 'qty': 5, 'symbol': 'DOLF20', 'time':
       '16:42:04.912363'}
21 {'event': 'SALE_OFFER_PARTIALLY_FILLED', 'id': '7545615515',
       'price': 4097.0, 'qty': 5, 'symbol': 'DOLF20', 'time':
       '16:42:04.912431'}
22 {'event': 'SALE_OFFER_PARTIALLY_FILLED', 'id': '7545615515',
       'price': 4097.0, 'qty': 5, 'symbol': 'DOLF20', 'time':
       '16:42:04.912500'}
23 {'event': 'SALE_OFFER_PARTIALLY_FILLED', 'id': '7545615515',
       'price': 4097.0, 'qty': 5, 'symbol': 'DOLF20', 'time':
       '16:42:04.912568'}
24 {'event': 'SALE_OFFER_PARTIALLY_FILLED', 'id': '7545615515',
       'price': 4097.0, 'qty': 5, 'symbol': 'DOLF20', 'time':
       '16:42:04.912636'}
25 {'event': 'SALE_OFFER_PARTIALLY_FILLED', 'id': '7545615515',
       'price': 4097.0, 'qty': 10, 'symbol': 'DOLF20', 'time':
       '16:42:04.912705'}
26 {'event': 'SALE_OFFER_ENTIRELY_FILLED', 'id': '7545615515',
       'price': 4097.0, 'qty': 5, 'symbol': 'DOLF20', 'time':
       '16:42:04.912774'}
27 {"id": "0400401321", "cash": 1003355.0, "profitAndLoss": 1.003355,
       "position": {"DOLF20": 0}}
```