UNIVERSIDADE FEDERAL DE ITAJUBÁ

PROGRAMA DE PÓS GRADUAÇÃO EM
CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

CronoSim - Uma Ferramenta Distribuída para Simulação de Eventos Discretos e Validação de Protocolos de Sincronização

Luiz Fernando Nunes

Itajubá, Junho de 2016

UNIVERSIDADE FEDERAL DE ITAJUBÁ

PROGRAMA DE PÓS GRADUAÇÃO EM
CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

Luiz Fernando Nunes

CronoSim - Uma Ferramenta Distribuída para Simulação de Eventos Discretos e Validação de Protocolos de Sincronização

Dissertação submetida ao Programa de Pós-Graduação em Ciência e Tecnologia da Computação como parte dos requisitos para obtenção do Título de Mestre em Ciência e Tecnologia da Computação

Área de Concentração: Matemática da Computação

Orientador: Prof. Dr. Edmilson Marmo Moreira

Coorientador: Prof. Dr. Otávio Augusto Salgado Carpinteiro

Junho de 2016
Itajubá - MG

UNIVERSIDADE FEDERAL DE ITAJUBÁ

PROGRAMA DE PÓS GRADUAÇÃO EM
CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

Luiz Fernando Nunes

CronoSim - Uma Ferramenta Distribuída para Simulação de Eventos Discretos e Validação de Protocolos de Sincronização

Dissertação aprovada por banca examinadora em 23 de Junho de 2016, conferindo ao autor o título de **Mestre em Ciência e Tecnologia da Computação**.

Banca Examinadora:

Prof. Dr. Edmilson Marmo Moreira (Orientador)
Prof. Dr. Otávio Augusto Salgado Carpinteiro (Coorientador)
Prof.ª Dr.ª Regina Helena Carlucci Santana (ICMC-USP)
Prof. Dr. Guilherme Sousa Bastos

Itajubá - MG
2016

Dedico este trabalho
aos meus pais Luiz e Célia,
e à minha esposa Patricia.

Agradecimentos

Agradeço a Deus que me abençoou em todos os momentos da minha vida, colocando pessoas boas em meu caminho.

Aos meus pais, Luiz e Célia, por serem exemplos de vida e, também, pelo incentivo e dedicação, os quais foram essenciais para minha educação e crescimento como pessoa. Sua motivação e compreensão foram fundamentais para que eu pudesse concluir esta etapa da minha vida. Agradeço a eles por sempre confiarem em mim e apoiarem minhas escolhas.

À minha esposa Patricia pelo amor, carinho, companheirismo e compreensão nos momentos de ausência. Seus conselhos e cobranças sempre me motivaram. Obrigado por cada dia ao meu lado.

Aos meus irmãos Reginaldo, Regiane, Rosiani e Thiago pelo companheirismo e amizade tão presente em nossa família.

Às minhas sobrinhas Letícia e Fernanda por trazerem alegria a todos a sua volta, inclusive, nos momentos em que mais precisamos.

A todos professores que estiveram comigo em algum momento da minha vida, por serem profissionais da educação, os quais valorizo muito e tenho um grande respeito.

Ao Prof. Edmilson, pela amizade, confiança, paciência, auxílio, conselhos e empenho ao longo do tempo que trabalhamos juntos. Agradeço, pois, foi a inspiração para que eu descobrisse minha vocação profissional, direcionando caminhos e oportunidades. Obrigado pela oportunidade de realizar este trabalho e por, sempre, me apoiar nos momentos decisivos da minha vida.

Ao Prof. Otávio Carpinteiro pela amizade, credibilidade e orientações nas conversas que tivemos.

Aos professores João Paulo Leite, Enzo Seraphim, Thatyana Seraphim e Bruno Kuehne pela amizade e pelo apoio em outras atividades que enriqueceram meu conhecimento.

Agradeço aos professores da UNIFEI, do IESTI e do IMC, que hoje são meus colegas de trabalho, por todo o conhecimento adquirido durante o curso de Bacharelado em Sistemas de Informação e do curso de Mestrado em Ciência e Tecnologia da Computação, que foi fundamental para a realização deste trabalho. Agradeço, também, aos colegas de mestrado, que se mostraram sempre disponíveis e contribuíram com sugestões e conselhos em relação ao trabalho.

Ao colega de mestrado Renan Curvello Faria pelo auxílio na compreensão sobre o projeto do *framework* e apoio na execução dos experimentos no *cluster* do GPESC.

Aos amigos da WILCX Desenvolvimento de Software Ltda, pelo apoio e flexibilidade em relação aos meus compromissos no início do mestrado.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio e incentivo a esta pesquisa.

Finalmente, agradeço a todas as pessoas que estiveram comigo no período de desenvolvimento da pesquisa.

Agradeço de todo coração a cada um!

"Não podemos prever o futuro, mas podemos criá-lo."
Peter Drucker

Resumo

Esta dissertação apresenta o projeto e implementação de uma ferramenta distribuída — CronoSim — para Simulação de Eventos Discretos. Ela fornece ao usuário um ambiente gráfico para auxiliá-lo durante a modelagem, execução e análise de resultados. CronoSim permite a simulação de forma distribuída, em que o modelo é dividido, e cada uma das partes é simulada paralelamente em processadores diferentes. Ela provê mecanismos para tratar, de maneira transparente ao usuário final, a sincronização da execução paralela de cada parte. Estes mecanismos foram baseados nos protocolos de sincronização *Time Warp* e *Rollback* Solidário, e utilizaram um padrão de comunicação por troca de mensagens. CronoSim também permite ao usuário analisar o desempenho de outros protocolos de sincronização. Desta forma, serve, também, como um ambiente de testes para pesquisadores na área de Simulação Distribuída, pois será disponível como um *software* de código aberto. Verificou-se que os resultados da simulação são gerados corretamente. Além disso, o desempenho de CronoSim foi avaliado experimentalmente e seus resultados são promissores.

Abstract

This work presents the design and implementation of a distributed tool — CronoSim — for Discrete Event Simulation. It provides the user with a graphical environment to assist him/her during the modeling, execution and analysis of the simulation. CronoSim allows the distributed simulation in which the model is divided, and each of its parts is simulated in parallel on different processors. It provides mechanisms to handle, in a transparent manner to the end user, the synchronization of the parallel execution of each part. These mechanisms are based on the synchronization protocols Time Warp and Solidary Rollback, and make use of a communication standard for exchanging messages. CronoSim also lets the end user analyze the performance of other synchronization protocols. Thus, it can be used, also, as a test environment for researchers in Distributed Simulation area, it will be available as an open source software. It was found that the simulation results are generated correctly. Furthermore, the performance of CronoSim was evaluated experimentally and its results are promising.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

AGPL	*Affero General Public License*
API	*Application Programming Interface*
AWT	*Abstract Window Toolkit*
BDS	*Berkeley Software Distribution*
FDAS	*Fixed Dependency After Send*
GPESC	Grupo de Pesquisas em Engenharia de Sistemas e de Computação
GPUs	*Graphics Processing Units*
GUI	*Graphical User Interface*
GVT	*Global Virtual Time*
IDE	*Integrated Development Environment*
IP	*Internet Protocol*
LVT	*Local Virtual Time*
MIT	*Massachusetts Institute of Technology*
MPI	*Message Passing Interface*
NFS	*Network File System*
PDES	*Parallel Discrete-Event Simulation*
PDF	*Portable Document Format*
PVM	*Parallel Virtual Machine*
RDT	*Rollback Dependency Trackability*
SED	Simulação de Eventos Discretos
SSS	*Sparse State Saving*
UML	*Unified Modeling Language*
XML	*eXtensible Markup Language*

1 Introdução

A simulação é uma técnica utilizada em projetos e avaliações de novos sistemas e também na reconfiguração física ou mudanças no controle de operação de sistemas existentes. Ela auxilia prever o comportamento de um sistema sem a sua existência real, antes que um possível investimento seja realizado. As suas aplicações têm crescido em todas as áreas, auxiliando os gestores na tomada de decisão em problemas complexos e melhorando o conhecimento dos processos nas organizações.

A simulação computacional possibilita analisar as dinâmicas de um processo e suas consequências. Diversos cenários podem ser criados e seus resultados verificados para auxiliar na tomada de decisão e, portanto, fornecer informações precisas de como o sistema reagiria com determinada modificação. A vantagem dessa operação é que a simulação visa repetir, a partir de um modelo, o mesmo comportamento que o processo real teria em condições semelhantes. O modelo de simulação computacional é utilizado, particularmente, como uma ferramenta para obter-se respostas a sentenças do tipo: *"o que ocorre se..."* (CHWIF; MEDINA, 2014).

Uma simulação pode ser classificada como determinística ou estocástica, de acordo com os dados fornecidos na entrada. A maioria dos sistemas a serem simulados são transformados em simulações estocásticas, onde os dados de entrada são gerados aleatoriamente a partir de uma distribuição de probabilidade. Para garantir que as variáveis estimadas minimizem a influência das variáveis aleatórias,

a simulação precisa ser repetida várias vezes (BRUSCHI, 2002). Como consequência disto, o tempo de execução da simulação se torna um fator desfavorável .

Neste contexto, abordagens paralelas foram adotadas para melhorar o desempenho e, assim, minimizar o tempo de execução da simulação. Neste caso, o sistema é particionado em processos lógicos e cada processo lógico é escalonado em um processador. A comunicação entre os processos é realizada através da troca de mensagens. No entanto, a paralelização da simulação cria dificuldades que não existem na simulação sequencial como, por exemplo: a sincronização entre os processos executados em diferentes processadores, o balanceamento de carga e a sobrecarga na rede de comunicação. Dessa forma, os conceitos de sincronização de processos levaram ao desenvolvimento de protocolos para garantir a sincronização entre os processos da simulação (FUJIMOTO, 2015).

1.1 Motivação

A realização de uma simulação, mesmo em um ambiente sequencial, é um trabalho que exige do usuário diversos conhecimentos como, por exemplo, modelagem, programação e matemática (probabilidade e estatística), tornando a criação, validação, execução e análise dos resultados obtidos uma atividade complexa para os usuários inexperientes.

Adotar uma das abordagens para paralelização da simulação é uma tarefa mais desgastante ainda. Além dos conhecimentos citados, necessita-se também de conhecimentos relacionados à Computação Paralela e Sistemas Distribuídos, que não são fáceis nem rápidos de se obter. Visto que o desenvolvimento de uma simulação distribuída necessita de um conjunto de conhecimentos que não é simples de se adquirir, é necessário uma solução para tornar esta atividade ao alcance dos usuários.

Neste sentido, a utilização de um ambiente de simulação, que apresente uma interface gráfica, suporte a execução distribuída de forma transparente e que per-

mita aos seus usuários maior agilidade no processo de criação de modelos e facilidade para execução da simulação em um ambiente distribuído é uma solução interessante.

Atualmente, vários ambientes e ferramentas de simulação são encontrados na literatura, sendo a maioria de uso comercial. Entretanto, o suporte à simulação distribuída com monitoramento de processos na rede de computadores onde é realizado o processamento, ainda, não é uma característica presente nas ferramentas.

O desenvolvimento de ferramentas para o monitoramento de processos permite que os recursos alocados nos nós de um sistema distribuído sejam monitorados e, eventualmente, reescalonados, a fim de melhorar o desempenho da simulação.

Portanto, estas características podem contribuir de maneira significativa para o uso da simulação, uma vez que trará vantagens relacionadas tanto ao processo de modelagem e análise dos resultados, quanto ao tempo de processamento.

Além disso, em uma simulação distribuída, onde os processos são particionados, é essencial a utilização de protocolos para sincronização. Diversos estudos vêm sendo realizados sobre este assunto, motivando adaptações nos protocolos existentes e, também, o desenvolvimento de novos protocolos, a fim de obter melhorias relacionadas à eficiência do processamento distribuído de uma simulação (FUJIMOTO, 2016). Em vista disto, pode-se notar que há necessidade de um ambiente para testes e validação de implementações correspondentes aos protocolos, uma vez que a análise sobre o desempenho realizada por meio de uma simulação distribuída em uma rede de computadores é uma abordagem mais natural.

1.2 Objetivos

Este trabalho de mestrado tem por objetivo oferecer aos usuários de simulação recursos que permitem tirar proveito do poder computacional de sistemas distribuídos na execução de simulações e, por conseguinte, obter melhorias no de-

sempenho. Em complemento, deseja-se apresentar à comunidade científica meios para validação e testes de protocolos para sincronização de processos para simulação distribuída.

Para este desiderato, esta dissertação apresenta o projeto e implementação de uma ferramenta para simulação distribuída de eventos discretos denominada CronoSim, que auxilia o usuário desde etapas como: modelagem, execução e análise de resultados até o monitoramento dos processos, tratando, principalmente, dos aspectos relacionados com o desempenho da simulação, através do balanceamento de carga do sistema e da possibilidade de controlar os mecanismos de migração de processos. A ferramenta também permite interromper a simulação em tempo de execução para verificar os resultados parciais e, também, verificar os estados dos processos lógicos do programa de simulação.

A utilização dos conceitos relacionados a sistemas distribuídos é realizada de forma transparente ao usuário. Desse modo, é possível simular modelos complexos sem se preocupar com detalhes específicos que este tipo de aplicação requer. O módulo de monitoramento da ferramenta possibilita ter conhecimento sobre os recursos utilizados pelos processos distribuídos, podendo intervir no escalonamento, caso ocorram situações de desbalanceamento.

A ferramenta CronoSim foi preparada para receber a implementação de novos protocolos de sincronização de processos para serem utilizados na execução da simulação distribuída. Neste sentido, outra característica que esta ferramenta possui é a possibilidade de analisar o desempenho desses protocolos. Com isto, é possível realizar comparações de protocolos utilizando modelos que podem ser desenvolvidos e executados por meio dos recursos oferecidos pela ferramenta.

1.3 Estrutura da Dissertação

Esta dissertação está organizada da seguinte forma: o próximo capítulo apresenta os conceitos envolvidos na área de Simulação e abrange uma introdução sobre

Simulação Distribuída e tipos de protocolos para sincronização de processos. São descritos, também, outras pesquisas acadêmicas relacionadas a este trabalho.

O capítulo 3 apresenta um estudo sobre diversas ferramentas disponíveis para simulação de eventos discretos, em especial, sobre aquelas que oferecem recursos para a Simulação Distribuída. Uma discussão sobre as características destas ferramentas é apresentada. Com este estudo, pôde-se levantar o estado da arte neste domínio, permitindo uma visão geral sobre das ferramentas existentes.

No capítulo 4, são apresentados os detalhes da ferramenta proposta neste trabalho. A arquitetura do projeto é descrita, bem como, a sua especificação apresentada através da UML (*Unified Modeling Language*).

O capítulo 5 abrange uma discussão sobre os resultados experimentais realizados, a fim de demostrar a eficácia da ferramenta proposta e as melhorias de desempenho obtidas devido à paralelização da simulação.

O último capítulo descreve as conclusões e contribuições sobre o trabalho, apresentando sugestões para trabalhos futuros.

2 Simulação e Computação Distribuída

A história da simulação pode ser apresentada de várias perspectivas, por exemplo, as formas de utilização e tipos de simulação, linguagens de programação utilizadas, ambientes de simulação e os domínios de aplicação (GOLDSMAN; NANCE; WILSON, 2010).

A simulação foi utilizada, inicialmente, nos EUA, no final da década de 1950, a fim de planejar operações militares como, distribuição de suprimentos nas batalhas e alocação de recursos escassos, obtendo resultados satisfatórios (HOLLOCKS, 2006).

A partir da década de 1970, a evolução e disseminação dos computadores contribuiu para o aumento da utilização da simulação e também permitiu a difusão em outras indústrias. Embora a maior parte das empresas que a utilizavam era de grande porte, muitas empresas menores passaram a ter grande interesse em utilizar a simulação no início de seus projetos (KELTON; SADOWSKI; STURROCK, 2014).

Na década de 1980, surgiram *softwares* de simulação com interfaces gráficas com o usuário, animação e outras ferramentas de visualização (BANKS et al., 2010). A simulação começou a ter maior utilização no início da década de 1990 por pequenas empresas (KELTON; SADOWSKI; STURROCK, 2014).

Segundo Harrell, Ghosh e Bowden (2004), a evolução da simulação é contínua e seu avanço é de acordo com o desenvolvimento de novas tecnologias de *softwares*,

possibilitando abstrair cada vez melhor os modelos e proporcionando melhores condições para análises e um amplo acesso aos tomadores de decisões.

Neste contexto, diversas definições são atribuídas à simulação e muitos autores na literatura a definem de forma distinta, porém, todas conduzem para a mesma ideia, o qual pode ser expressa como sendo a execução e análise de resultados de modelos.

De acordo com Banks et al. (2010), uma simulação pode ser definida como uma imitação de um processo do mundo real sobre o tempo, caracterizada pela geração de um histórico artificial do comportamento de algum sistema. A análise deste histórico possibilita criar deduções relacionadas às características operacionais, analisar o comportamento e elaborar questões sobre o sistema real simulado.

Para Montevechi et al. (2007), a simulação é a importação da realidade para um ambiente controlado, sendo possível analisar seu comportamento sob diversas condições, sem riscos físicos e/ou altos custos envolvidos. Nesse sentido, a realidade pode ser representada por meio da utilização de modelos, buscando experimentar as possíveis alternativas para conseguir tomar a melhor decisão.

Um modelo é uma abstração da realidade de modo a reproduzir o verdadeiro comportamento do sistema, porém, deve ser mais simples do que o sistema real. Modelos devem ser complexos o suficiente para responder as questões levantadas, mas devem apresentar uma complexidade menor do que o sistema real, pois, caso contrário, obtêm-se um problema e não um modelo propriamente dito (BANKS et al., 2010; CHWIF; MEDINA, 2014).

Sakurada e Miyake (2009) explicam que a grande variedade de *softwares* de simulação disponíveis no mercado favorece a aplicação da simulação de uma forma geral. A competitividade entre as empresas fabricantes de *softwares* de simulação tem impulsionado o desenvolvimento de novas facilidades tais como ferramentas de suporte ao processo de modelagem, recursos de análise estatística e interfaces gráficas intuitivas (*user-friendly*).

2.1 Tipos de Simulação

Segundo Harrell, Ghosh e Bowden (2004), um modelo de simulação engloba quatro elementos principais:

- **Entidades**: itens processados ao longo do sistema, como produtos, clientes e documentos;
- **Atividades**: tarefas que são executadas no sistema relacionadas de forma direta ou indireta ao processamento de entidades;
- **Recursos**: meios pelos quais as atividades são realizadas, como instalações, equipamentos e pessoas;
- **Controle**: parâmetros que ditam como, quando e onde as atividades são realizadas. São as regras do sistema.

Ainda é possível citar outros elementos como: filas, entradas e saídas. Utilizando este novos itens, Chwif e Medina (2014) exemplificam suas aplicações dizendo que "**entidades chegam** ao sistema, esperam ou ficam armazenadas em filas, são processadas em **atividades** (que podem utilizar **recursos**) e depois **saem** do sistema".

Chwif e Medina (2014) classificam as simulações em computacionais e não computacionais, sendo a simulação computacional dividida em três categorias básicas: Simulação de "Monte Carlo", Contínua e de Eventos Discretos. A simulação de Monte Carlo é baseada em geradores de números aleatórios para simular sistemas físicos e matemáticos, nos quais o tempo não é considerado explicitamente como uma variável. A Simulação Contínua e a Simulação de Eventos Discretos (SED) consideram a mudança de estado do sistema no decorrer do tempo, mas se diferem pelo fato de que a primeira é utilizada em sistemas cujo estado varia no tempo de forma contínua, ao passo que, na segunda, o estado varia em momentos discretos do tempo, a partir da ocorrência de eventos.

Observando por um ponto de vista mais amplo, um modelo de simulação pode ser classificado a partir de vários aspectos. Os itens a seguir apresentam a classificação dos modelos:

1. **Quanto ao tempo**
    - **Estático**: o tempo não é considerado (simulação de Monte Carlo);
    - **Dinâmico**: sistemas que alteram seu estado ao longo do tempo.
2. **Quanto a sua aleatoriedade**
    - **Determinístico:** os valores considerados são constantes;
    - **Estocástico**: os valores de entrada do modelo são aleatórios e seguem uma distribuição de probabilidade.
3. **Quanto ao estado de mudança**
    - **Contínuo**: quando o estado do sistema muda continuamente no tempo;
    - **Eventos discretos**: o estado se altera a partir da ocorrência de um evento no decorrer do tempo.

A figura 1 apresenta uma taxonomia relacionando todas as classificações de tipos de modelos de simulação apresentados. A ferramenta desenvolvida neste trabalho considera modelos classificados como estocástico, dinâmico e a eventos discretos.

2.2 Simulação Paralela e Distribuída

Conforme apresentado nas seções anteriores, a simulação utilizada para avaliar o comportamento de um sistema é uma poderosa ferramenta. No entanto, um programa de simulação executando sobre um único processador pode levar muito

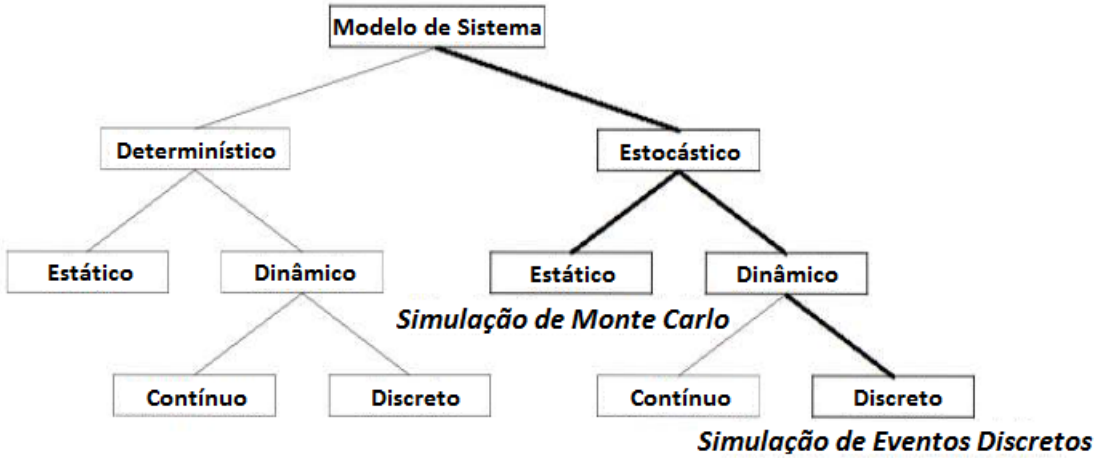

Figura 1: Taxonomia de modelos de simulação (BABULAK; WANG, 2010)

tempo para apresentar um resultado confiável quando os modelo são mais complexos, com mais detalhes. Devido à aleatoriedade das variáveis de entrada, o programa deve ser executado diversas vezes para que não haja interferência nos resultados da simulação. Desta forma, o conceito de simulação paralela e distribuída vem sendo utilizado para reduzir o tempo de execução da simulação. Esta área é também conhecida como PDES (*Parallel Discrete Event Simulation*).

O termo "simulação paralela de eventos discretos" (*PDES - Parallel Discrete-Event Simulation*) vem sendo utilizado em referência a execução dos programas de simulação em plataformas computacionais de múltiplos processadores (FUJIMOTO, 2016). De uma forma específica, a expressão "simulação paralela" refere-se ao processamento da simulação em um sistema fortemente acoplado, como, por exemplo, um multiprocessador de memória compartilhada. Por outro lado, o termo "simulação distribuída" está relacionado com a exploração de plataformas de computação distribuída que podem abranger uma extensão geográfica mais ampla, que vão desde máquinas interligadas através de uma rede local até computadores distribuídos globalmente e que se comunicam através da internet (YILMAZ et al., 2014).

Com o desenvolvimento das tecnologias nesta área e surgimento de novos para-

digmas computacionais, como "Computação em *Cluster*" e "Computação em Nuvem", o conceito de simulação paralela e distribuída tem evoluído (D'ANGELO, 2011; ZEHE et al., 2015). A utilização de uma infraestrutura computacional robusta permite otimizar o processamento de forma considerável. Por exemplo, a simulação aliada à Computação em Nuvem traz mais transparência à execução, pois, este paradigma propõe a computação como um serviço e, dessa forma, os serviços de infraestrutura (*IaaS - Infrastructure as a Service*), plataforma (*PaaS - Platform as a Service*) e *software* (*SaaS - Software as a Service*) são atribuídos à nuvem (FUJIMOTO; MALIK; PARK, 2010).

Na simulação distribuída o modelo de simulação é dividido e cada uma das partes é simulada em um processador diferente. A utilização de diversos processadores requer uma adaptação em relação à simulação sequencial. Desenvolver programas de simulação de eventos discretos com processamento distribuído não é uma tarefa simples, visto que é necessário tomar cuidados específicos com a estrutura da simulação. No paradigma da simulação de eventos discretos, três estruturas de dados são utilizadas para a implementação das primitivas básicas no desenvolvimento da simulação:

1. As variáveis de estado que armazenam o estado do sistema;
2. Uma lista de eventos futuros, ou seja, com eventos que estão aguardando em uma fila para serem processados;
3. Um relógio global que controla progresso da simulação.

Em uma simulação discreta, cada evento contém uma marca de tempo (*timestamp*) que representa quando uma mudança deve ocorrer no sistema. A lista de eventos futuros é classificada em ordem crescente ao tempo de chegada. Durante a execução da simulação, uma estrutura de repetição faz com que o evento com menor *timestamp* seja retirado da lista e processado. Processar um evento consiste em executar algum código para efetuar a mudança de estado e, se necessário,

gerar novos eventos para o futuro. Quando o evento é retirado da lista, o relógio global é, então, avançado para o tempo de ocorrência do evento e o programa de simulação inicia a sua execução. Um detalhe importante nesta abordagem é que o evento com menor *timestamp* (E_{mim}) deve ser o próximo evento a ser processado. Isto garante que os eventos sejam simulados em ordem cronológica no tempo de simulação.

Na implementação da simulação distribuída, o sistema passa a ser dividido em um conjunto de n processos lógicos p_1, p_2, ..., p_n, cada um representando um processo do sistema real. Todavia, esta nova abordagem encontra obstáculos não existentes em um sistema centralizado, tais como, sincronização dos processos, balanceamento de carga e a sobrecarga na rede de comunicação. Em um ambiente de simulação distribuída, os processos lógicos devem ser executados de maneira que se evite a ocorrência de erros de causa e efeito, ou seja, a execução de eventos fora da ordem natural que seria obtida em um sistema centralizado. Para que isso ocorra, cada processo lógico tem seu próprio relógio lógico, que indica o progresso da simulação e, como não há um ambiente de memória compartilhada entre eles, as informações entre os processos são trocadas através de mensagens (YILMAZ et al., 2014).

Neste sentido, uma simulação distribuída, para ter validade, requer a não ocorrência de erros de causa e efeito. Para isto, a relação de precedência causal deve ser respeitada pelos eventos. A relação de ordem pode ser sintetizada da seguinte maneira: dado dois eventos, e_1 e e_2, então e_1 ocorre antes de e_2 se o *timestamp* de e_1 for menor que o *timestamp* de e_2.

Uma maneira de garantir a ordem de execução cronológica dos eventos é utilizando protocolos que controlem o sincronismo entre os processos da simulação. Os protocolos de sincronismo que foram desenvolvidos são divididos em duas classes: conservativos e otimistas, evitando ou corrigindo erros de causa e efeito, respectivamente (FUJIMOTO, 2015).

A principal característica de um protocolo conservativo é a determinação de

quando é seguro executar, ou seja, os tempos de ocorrência dos eventos devem ser verificados para que nenhum deles seja executado fora da sua ordem cronológica. Os protocolos conservativos evitam que erros de causalidade ocorram durante a execução da simulação, estabelecendo, portanto, uma sincronização explícita.

Por outro lado, os protocolos otimistas permitem que os processos executem eventos fora de sua ordem cronológica, fornecendo uma maneira de corrigir o problema quando ocorre. Dessa forma, se um erro for detectado, os protocolos otimistas possuem mecanismos para recuperar o sistema e manter a sua consistência (JAFER; LIU; WAINER, 2013). A vantagem neste tipo de protocolo é a possibilidade de aproveitar melhor o paralelismo em situações onde erros poderiam ocorrer, mas não ocorrem (MALIK; PARK; FUJIMOTO, 2010). A próxima subseção apresenta as principais características deste tipo de protocolo, visto que o desenvolvimento deste trabalho considerou a implementação de dois protocolos otimistas, que serão explicados em sequência.

2.3 Protocolos Otimistas

Nos protocolos otimistas, para sincronizar os processos e manter o controle cronológico do sistema, são utilizadas as seguintes informações:

- *Local Virtual Time* (LVT) de cada processo que permite acompanhar a ordem cronológica de sua computação.
- *Timestamp* de cada evento, com a função de indicar quando o evento deve ser executado;
- *Global Virtual Time* (GVT) que representa o menor LVT do sistema, estabelecendo que nenhum processo do sistema irá criar um evento com o *timestamp* menor do que o valor do GVT.

O termo otimista está relacionado com a execução dos eventos pelos processos

lógicos de modo otimista presumindo que não irão ocorrer erros de causa e efeito. Assim como os protocolos conservativos, não existem estados compartilhados. A comunicação é exclusivamente realizada por troca de mensagens rotuladas com o *timestamp* e através de canais confiáveis, ou seja, toda mensagem chega ao destino. Dentre as diferenças apresentadas por esse tipo de protocolo em relação aos protocolos conservativos, pode-se mencionar que, neste caso, os processos lógicos não necessitam enviar as mensagens na ordem de seus rótulos e os canais de comunicação não garantem a entrega das mensagens na mesma ordem em que elas foram enviadas. Além disso, durante a execução da simulação, os processos podem ser criados e destruídos e não é necessário definir, explicitamente, quais processos lógicos podem comunicar entre si.

Esta liberdade que os processos possuem aumenta a possibilidade de ocorrer erros de causa e efeito. Por conseguinte, quando um erro é detectado, um mecanismo é utilizado para recuperar o sistema e voltar a um estado seguro. Este mecanismo é conhecido como *rollback*, devido ao fato de ser restaurado o estado imediatamente anterior ao tempo lógico da mensagem quando um processo recebe uma mensagem com marca de tempo menor que o seu LVT. A mensagem que ativa o *rollback* é denominada *straggler*. Para isso, é necessário que os estados dos processos sejam salvos periodicamente na lista de estados.

Se determinado processo, que está realizando um *rollback*, enviou mensagens para outros processos com tempo lógico maior que o tempo para onde ele está retornando, estas mensagens precisam ser canceladas. Desta maneira, o processo envia "antimensagens" avisando do cancelamento dos respectivos eventos (MOREIRA, 2005).

Dois fatores podem implicar em um baixo desempenho do programa de simulação. O primeiro está relacionado com a ocorrência de *rollback*, devido ao tempo de processamento perdido na execução dos eventos desfeitos, além da necessidade de avisar os processos que recebem suas mensagens quando desfaz parte do seu processamento, o que pode produzir novos *rollbacks*. O segundo fator é a ne-

cessidade de armazenar, continuamente, os estados de cada processo para futura recuperação, o que degrada o sistema.

Devido à forma como é realizada a reconstituição da computação, a partir do surgimento de uma mensagem *straggler*, podem ser identificados dois tipos de *rollbacks* nos protocolos otimistas tradicionais: primários e secundários. *Rollback* primário ocorre quando um processo necessita realizar um *rollback* através do envio de uma mensagem *straggler*. Quando um processo necessita realizar um *rollback* devido a uma antimensagem, este *rollback* é denominado *rollback* secundário. A ocorrência de um *rollback* pode provocar vários *rollbacks* secundários ou *rollbacks* em cascata. Desta forma, percebe-se que a sincronização dos processos acontece lentamente, permitindo que aqueles que sofrerão *rollbacks* secundários continuem a sua computação, mesmo que a mensagem *straggler* já tenha sido recebida pelo processo lógico destinatário e esse já esteja recuperando o seu estado (MOREIRA, 2005).

Jefferson (1985) desenvolveu um protocolo denominado *Time Warp*, sendo este, o protocolo otimista mais conhecido e explorado na literatura. Os conceitos elementares e mecanismos empregados em outros protocolos otimistas tais como *rollback*, antimensagens e controle global do tempo de simulação, apareceram, a princípio, no *Time Warp* (FUJIMOTO, 2000). Diversas variações foram propostas a fim de melhorar o desempenho dos protocolos otimistas. Moreira (2005) propôs outro protocolo otimista denominado *Rollback* Solidário, que utiliza o conceito de *checkpoints* globais consistentes para melhorar a sincronização dos processos durante o procedimento de *rollback*.

As subseções seguintes descrevem as principais características dos protocolos *Time Warp* e *Rollback* Solidário, respectivamente, uma vez que estes protocolos foram utilizados no desenvolvimento da ferramenta proposta neste trabalho de mestrado.

2.3.1 Time Warp

O protocolo *Time Warp* se baseia no fato de que nenhum erro de causa e efeito irá ocorrer no futuro. Com essa proposição, os processos lógicos executam os seus eventos sem observar o andamento dos demais. No entanto, quando um evento fora de ordem é detectado, ou seja, uma mensagem *straggler* é recebida, o processo que a identificou realiza *rollback* imediatamente, para que o sistema se mantenha consistente.

A figura 2 ilustra a chegada de uma mensagem *straggler* no processo. Os eventos 12, 21 e 35 já foram processados, então, o tempo lógico do processo é igual a 35. Sendo assim, o próximo evento a ser tratado é o de *timestamp* igual a 41. Neste instante, uma mensagem com *timestamp* igual a 18 chega ao processo, produzindo um erro de causalidade, visto que ela deveria ter sido processada antes dos eventos 21 e 35. Dessa forma, o sistema deve realizar *rollback*, retornando a computação para o estado imediatamente após a execução da mensagem 12, processar a mensagem 18 e continuar a partir deste ponto. Os eventos com *timestamp* 21 e 35 são colocados novamente na fila de eventos futuros e deverão ser reprocessados no tempo lógico correto.

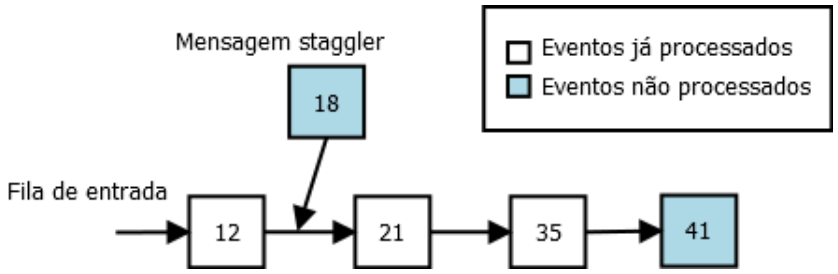

Figura 2: Eventos de um processo lógico durante a chegada de uma mensagem *straggler* (FUJIMOTO, 2000)

São utilizados dois mecanismos de recuperação da simulação: os mecanismos de controle local e os de controle global. A implementação do mecanismo de controle local é realizada dentro de cada processo lógico de forma independente dos outros processos, garantindo que os eventos sejam processados em ordem correta.

O mecanismo de controle global é responsável pelo gerenciamento de memória e pelo cálculo do *Global Virtual Time* (GVT). O GVT se baseia no menor valor entre todos os LVTs dos processos lógicos e das mensagens em trânsito. Após o cálculo do GVT, todas as informações de estados anteriores a este tempo podem ser descartadas, pois é garantido que até este tempo não é possível ocorrer qualquer erro de causa e efeito (JEFFERSON, 1985; FUJIMOTO, 2015).

Para cada mensagem que foi enviada para outros processos após a ocorrência de um *rollback*, deve ser realizado o seu cancelamento. Isto deve ser feito através do envio de uma antimensagem correspondente. Caso o processo receba uma antimensagem cujo evento ainda não foi processado, ele remove a mensagem da lista de eventos futuros. Neste contexto, três estratégias de cancelamento foram desenvolvidas: cancelamento agressivo, cancelamento preguiçoso e cancelamento dinâmico (FUJIMOTO, 1990).

1. **Cancelamento agressivo:** as antimensagens são imediatamente enviadas para desfazer todas as mensagens anteriormente submetidas após um processo efetuar um *rollback*.

2. **Cancelamento preguiçoso:** as antimensagens não são enviadas imediatamente após a ocorrência de um *rollback*. Neste caso, esperam para verificar se a nova execução da computação produz os mesmos eventos, possibilitando identificar se é necessário efetuar o cancelamento.

3. **Cancelamento Dinâmico:** utiliza-se uma técnica em que cada processo decide qual das duas estratégias anteriores deve utilizar.

Outro aspecto importante é a necessidade do armazenamento das variáveis de estado para garantir a possibilidade de restaurar a computação em caso de falhas. Para isto, existem mecanismos de salvamento de estados que possibilitam a sua recuperação em caso de *rollback*, podendo assim, serem gerados novamente. Para garantir que os estados possam ser recuperados em caso de *rollback*, o sistema

deve armazenar informações suficientes a respeito dos processos. Desta forma, o mecanismo de salvamento de estados deve ser bem projetado uma vez que ele interfere no desempenho do sistema, além do gasto de memória (LAPRE et al., 2015).

O método mais simples de gerenciamento de memória denomina-se *copy state saving*, o qual armazena todos os estados sempre que um evento é executado. Entretanto, essa estratégia além de exigir alta sobrecarga do processador, também requer muito espaço de armazenamento. Com objetivo de reduzir esta sobrecarga, foram propostas outras estratégias que se classificam em duas categorias: *sparse state saving* e *incremental state saving*. Na utilização do *sparse state saving*, os estados são salvos periodicamente e não mais a cada execução de evento. Já no caso do *incremental state saving*, são salvos somente os que foram modificados nos estados a cada execução do evento (RONNGREN et al., 1996; FUJIMOTO, 2000).

Apesar da utilização de *rollbacks* permitir grande liberdade na execução de eventos nos protocolos otimistas, o que proporciona o aumento do desempenho na simulação, existe a desvantagem relacionada com a necessidade de armazenamento de antimensagens que ocupam grande espaço na memória do sistema. Neste contexto, Moreira (2005) propôs um protocolo otimista que está fundamentado na teoria dos *Checkpoints* Globais Consistentes (MOREIRA et al., 2005), utilizada em sistemas tolerantes a falhas, produzindo uma abordagem diferente para o sincronismo dos processos durante um *rollback* em relação aos demais protocolos otimistas que alcançam o sincronismo através das antimensagens. A subseção seguinte apresenta algumas características do protocolo otimista *Rollback* Solidário.

2.3.2 Rollback Solidário

O protocolo *Rollback* Solidário, assim como o protocolo *Time Warp*, também utiliza *rollbacks* para recuperar estados quando um erro de causa e efeito ocorre. Esta e outras características fazem que ele pertença à classe de protocolos otimistas, porém, há diferenças significativas que não transformam o protocolo *Rollback* Solidário em uma variante do protocolo *Time Warp* (CRUZ, 2009)

O princípio de funcionamento do protocolo *Rollback* Solidário surge da constatação de que, utilizando *checkpoints* globais consistentes, é possível retornar todos os processos que estarão envolvidos no *rollback* para um *checkpoint* global consistente, durante a ocorrência de algum erro de causa e efeito. Assim sendo, a decisão do retorno é tomada com base nas informações de todos os processos. O nome *Rollback* Solidário se originou dessa ideia, ou seja, quando um processo precisa retornar, os demais "ajudam" na solução do problema e, se necessário, retornam "juntos" (MOREIRA, 2005).

Como foi apresentado na subseção anterior, no protocolo *Time Warp*, quando um processo identifica uma mensagem *straggler*, é realizado imediatamente o *rollback* para manter a consistência do sistema. Uma vez que o processo recupera sua computação, é necessário identificar todas as mensagens que foram enviadas durante este período de tempo e enviar uma antimensagem a cada processo que recebeu alguma mensagem para que eles corrijam seus estados.

No protocolo *Rollback* Solidário, diferente do mecanismo utilizado no *Time Warp*, não são utilizadas antimensagens e, portanto, não é preciso manter cópia das mensagens trocadas entre os processos. Quando surge uma mensagem *straggler*, são identificados simultaneamente os estados que devem ser recuperados pelos processos que estão envolvidos no *rollback*. Ou seja, retornam todos os processos que estarão envolvidos em um *rollback* para um *checkpoint* global consistente, garantindo que não irão ocorrer *rollbacks* em cascata, além de facilitar o cálculo do GVT. Neste contexto, podem ser observadas outras vantagens relacionadas com a eliminação das antimensagens, como a diminuição do tráfego na rede de comunicação, pois não há antimensagens ocupando os canais de comunicação (MOREIRA, 2005).

São utilizados dois tipos de *checkpoints*: básicos e forçados. No caso dos *checkpoints* básicos, o mecanismo de salvamento de estado no protocolo *Rollback* Solidário é similar ao mecanismo *sparse state saving* utilizado pelo protocolo *Time Warp*. Os estados são salvo em um intervalo determinado de tempo ou de eventos.

O salvamento de *checkpoints* forçados se baseia no algoritmo *Fixed Dependency After Send* (FDAS). Uma característica importante deste algoritmo é a implementação do padrão de *checkpoints* que satisfaz a propriedade *Rollback Dependency Trackability* (RDT), em que todas as dependências entre *checkpoints* podem ser rastreadas em tempo de execução através da utilização de vetores de dependências ou relógios vetoriais (WANG, 1997; RAYNAL, 2013).

2.4 Um Framework Para Simulação Distribuída

As seções anteriores apresentaram os conceitos sobre protocolos de sincronização utilizados no desenvolvimento de aplicações paralelas e/ou distribuídas. Esta seção descreve um *framework* que propicia o desenvolvimento de programas de simulação capazes de executar em uma infraestrutura computacional distribuída. A especificação pertinente a este *framework* envolve os conceitos explicados anteriormente, uma vez que os protocolos *Time Warp* e *Rollback* Solidário são implementados.

Um *framework* orientado a objetos é um conjunto de classes e interfaces que incorpora um projeto abstrato. Ele provê uma infraestrutura genérica para construção de aplicações dentro de uma família de problemas semelhantes, de forma que esta infraestrutura genérica deve ser adaptada para a geração de uma aplicação específica. O conjunto de classes que forma o *framework* deve ser flexível e extensível para permitir a construção de várias aplicações com pouco esforço, especificando apenas as particularidades de cada aplicação (ONISHI, 2006).

A reutilização de partes de soluções de um programa já desenvolvido em outros programas é uma característica da programação orientada a objetos, o que reduz o custo no desenvolvimento de um novo programa. A partir dessa prática, foram criadas várias classes que possibilitam a reutilização de soluções prontas, que são caracterizadas como bibliotecas. Entretanto, com o desenvolvimento de *frameworks*, esta prática se tornou mais completa, devido ao fato de levar a imple-

mentação de ferramentas através de componentes que podem ser utilizados sem a necessidade de conhecer o seu código original. A diferença entre estes dois conceitos está relacionada com o modo de utilização, em que no caso do *framework* todas as associações, generalizações, dependências e refinamentos já estão desenvolvidos dentro do componente, enquanto que em uma biblioteca de classes são oferecidas apenas as classes desenvolvidas e todo fluxo de controle deve ser implementado pelo usuário (JOHNSON; FOOTE, 1988; JALENDER; GOVARDHAN; PREMCHAND, 2012)

2.4.1 Vantagens na Utilização de Frameworks

Enquanto a utilização de *frameworks* reduz esforços e aumenta a qualidade de *softwares*, existe uma série de desafios que devem ser dominados para que eles possam ser utilizados com maior eficiência e eficácia. O primeiro desafio é escolher qual *framework* é mais apropriado para a linha de pesquisa ou desenvolvimento da instituição ou empresa que irá empregá-lo. Uma escolha imprópria pode comprometer todo o projeto invalidando todos os benefícios e reduzindo os custos esperados na aquisição de um *framework* (CRUZ, 2009). O próximo passo é adquirir o conhecimento da ferramenta adotada, a fim de utilizar todos os recursos disponíveis adequadamente.

Cruz (2009) cita algumas vantagens na utilização de *frameworks*, tais como:

- Preservação de conhecimentos, devido ao desenvolvimento a partir de soluções de problemas específicos encontradas por vários programadores.
- Maior velocidade e menor custo no desenvolvimento de novos *softwares*, pois várias funcionalidades já estão prontas;
- Soluções confiáveis, robustas e de alta qualidade, pois são testadas por muitos programadores;
- Quando erros são descobertos, são eliminados e novas funcionalidades podem ser incrementadas, adquirindo, assim, certa maturidade;

- Manutenção é rápida e fácil, pois para corrigir ou criar alguma funcionalidade, geralmente não há a necessidade de se alterar todo o código do *framework*, apenas as partes relacionadas.

2.4.2 Descrição do Framework Utilizado

Existem diversas ferramentas e *frameworks* para o desenvolvimento de aplicações distribuídas. Todavia, essas ferramentas não são específicas para aplicações de simulação, exigindo dos usuários, além dos conhecimentos específicos de simulação, a necessidade de implementar diversos recursos extras para que se possa desenvolver um programa para simulação. Neste contexto, Cruz (2009) propôs um *framework* que oferece suporte ao desenvolvimento de programas de simulação distribuída (MOREIRA et al., 2010).

Nesta subseção são apresentadas as principais características deste *framework*, bem como seus recursos disponíveis, uma vez que utiliza os protocolos que formam o eixo deste trabalho de mestrado. Outros trabalhos, também, o adotaram como base para o desenvolvimento de novos mecanismos relacionados aos protocolos otimistas, além de oferecer adaptações e melhorias ao próprio *framework*.

Neste sentido, Azevedo (2012) apresenta uma variação deste *framework* com implementações dos protocolos *Time Warp* e *Rollback* Solidário utilizando a plataforma Java. Tal alternativa foi inicialmente baseada no *framework* original, porém reformula as classes que manipulam os processos envolvidos na simulação, o protocolo de sincronização e o mecanismo de salvamento de estados. Segundo o autor, o fato do novo *framework* ser implementado em tecnologia Java traz algumas vantagens estratégicas. Entre elas, a utilização de reflexão, que envolve, entre outros conceitos, o carregamento de classes em tempo de execução, além de permitir a agregação do amplo leque de tecnologias que orbitam a tecnologia Java.

Fundamentado neste último trabalho, recentemente, Faria (2016) propôs um método para análise de troca de protocolos otimistas em ambiente de simulação

distribuída e também apresentou melhorias em relação às implementações anteriores do *framework*. A ideia consiste em determinar quando é vantajoso efetuar a troca dinâmica entre os dois protocolos em questão, através da análise e comparação dos resultados entre intervalos de tempo previamente definidos do total de tempo da execução da simulação.

Para melhor compreensão do *framework* de Cruz (2009) e as melhorias propostas por Azevedo (2012) e Faria (2016), as subseções seguintes descrevem os detalhes inerentes a cada trabalho. Estes três estudos foram desenvolvidos pelo Grupo de Pesquisas em Engenharia de Sistemas e de Computação (GPESC) da Universidade Federal de Itajubá. É importante destacar que este detalhamento é necessário para a compreensão da ferramenta desenvolvida neste trabalho de mestrado, uma vez que formam a base para seu desenvolvimento.

2.4.2.1 Framework Proposto por Cruz (2009)

Conforme explicado anteriormente, o projeto das classes deste *framework* considerou dois protocolos de sincronização otimistas: *Time Warp* e *Rollback* Solidário. A figura 3 mostra o esquema do *framework*.

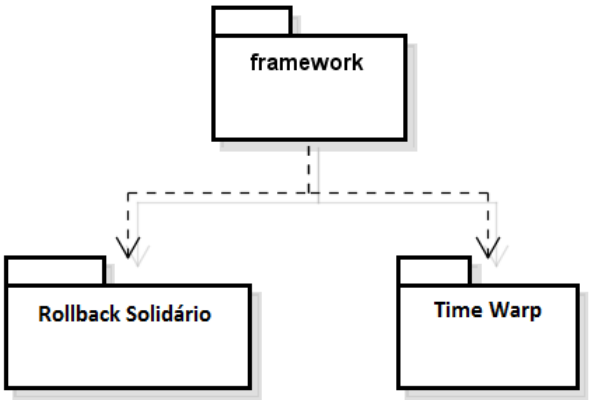

Figura 3: Representação esquemática do *framework*

Parte dos recursos disponíveis está relacionada à possibilidade de utilizar diferentes tipos de protocolos de sincronização em uma mesma aplicação e utilizar

bibliotecas de comunicação diferentes, como o PVM (*Parallel Virtual Machine*) e o MPI (*Message Passing Interface*). Além disso, é utilizado o modelo em camadas, como pode ser observado na figura 4, o que possibilita um alto nível de flexibilidade, permitindo combinar diversas estruturas de forma simples e eficiente (MOREIRA et al., 2010).

Figura 4: Estrutura de um ambiente de Simulação Distribuída (CRUZ, 2009)

Cruz (2009), ao estruturar as camadas, definiu que o primeiro nível, formado pela camada **Arquitetura Física**, é responsável em manter as informações atualizadas da arquitetura física onde o programa de simulação será executado. O segundo nível, formado pela camada **Comunicação**, é responsável pelas trocas de informações realizadas durante a simulação. Estas trocas de informações são feitas através do envio e do recebimento de mensagens entre os processos envolvidos na simulação, através dos canais de comunicação do sistema. Os algoritmos implementados nesta camada também garantem que toda mensagem enviada por um processo será recebida pelo processo receptor e que a ordem cronológica de envio será respeitada no recebimento.

O terceiro nível é formado pela camada de **Sincronização**, sendo responsável por garantir a consistência dos resultados obtidos pela simulação, tratando, assim, os possíveis erros de causa e efeito e garantindo que a restrição de causalidade local de cada processo será respeitada.

Por fim, a camada de **Aplicação** é onde os modelos de simulação, providos como entrada para o *framework*, serão efetivamente executados e onde os dados serão coletados e entregues em uma saída definida.

O diagrama de classes da figura 5 apresenta a estrutura básica do *framework*.

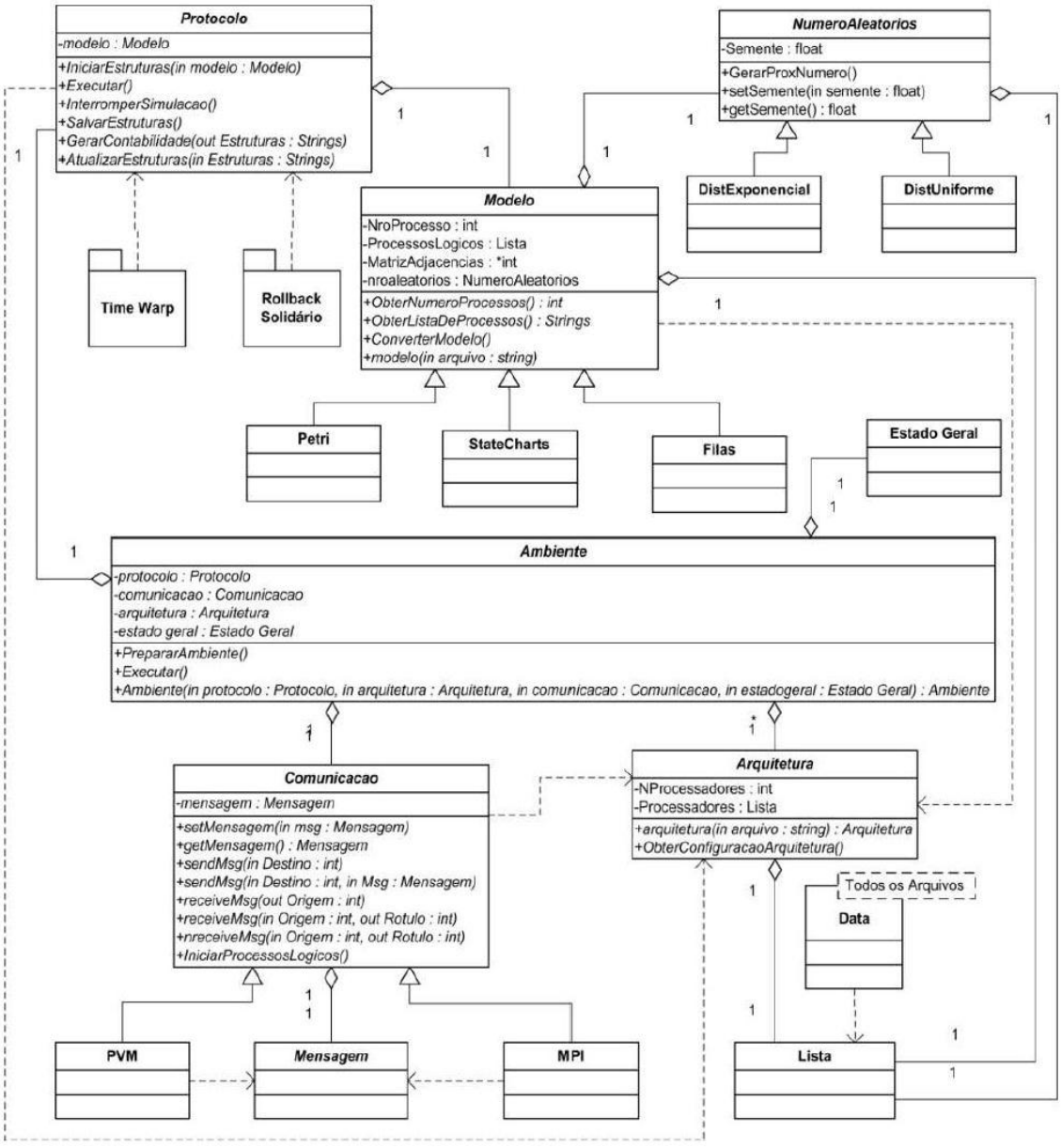

Figura 5: Diagrama de Classes do *framework* proposto por Cruz (2009)

Cruz (2009) apresentou, também, os diagramas de classes para as implementações dos protocolos *Rollback* Solidário (figura 6) e *Time Warp* (Figura 7). Em ambos os diagramas, cada processo se inicia ao chamar o método **executar()** da classe **Processo**, que possui implementações nas classes concretas **ProcessoComObservador** e **Observador**.

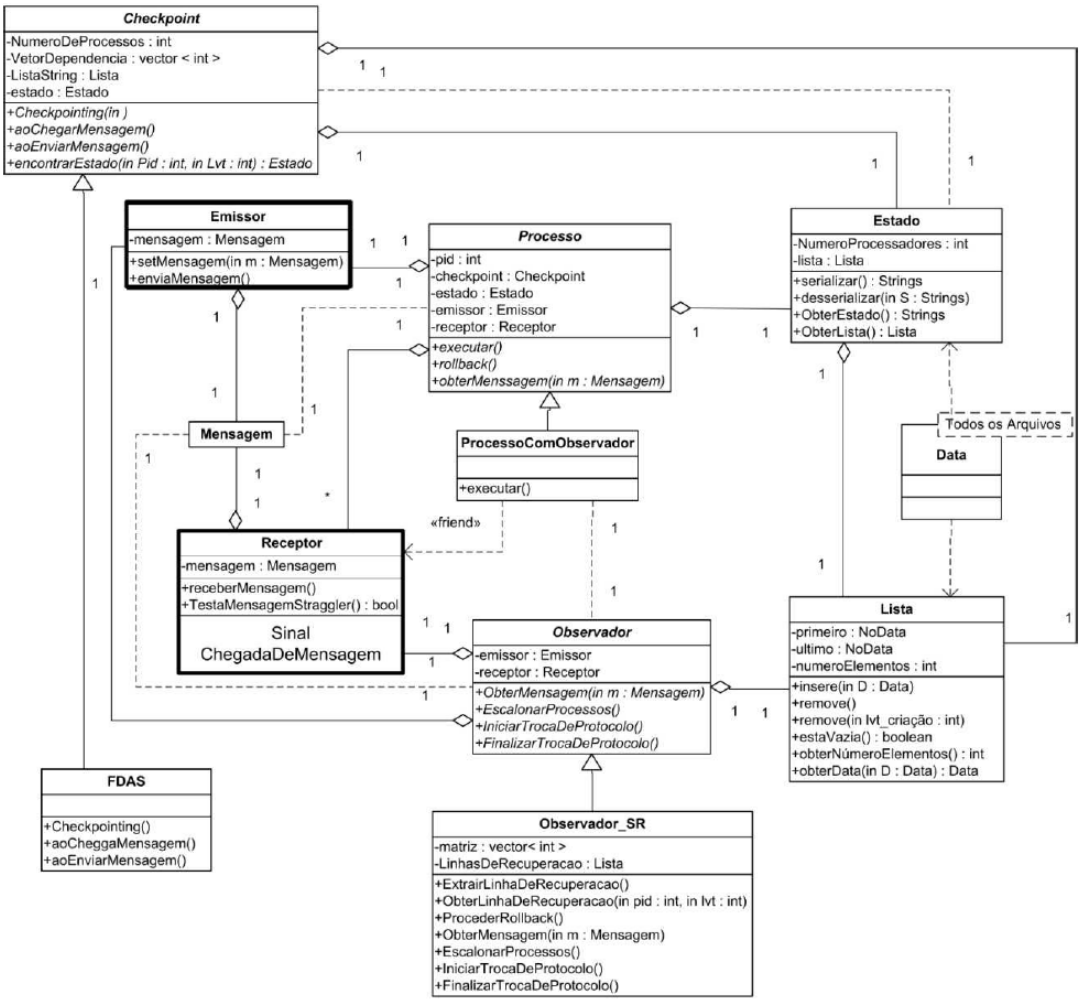

Figura 6: Diagrama de classes proposto para o *Rollback* Solidário (CRUZ, 2009)

O diagrama da figura 6 foi, inicialmente, apresentado por Moreira (2005) como proposta de implementação para o *Rollback* Solidário. Tal implementação foi integrada ao *framework* em questão. A classe **Checkpoint** é utilizada para marcar

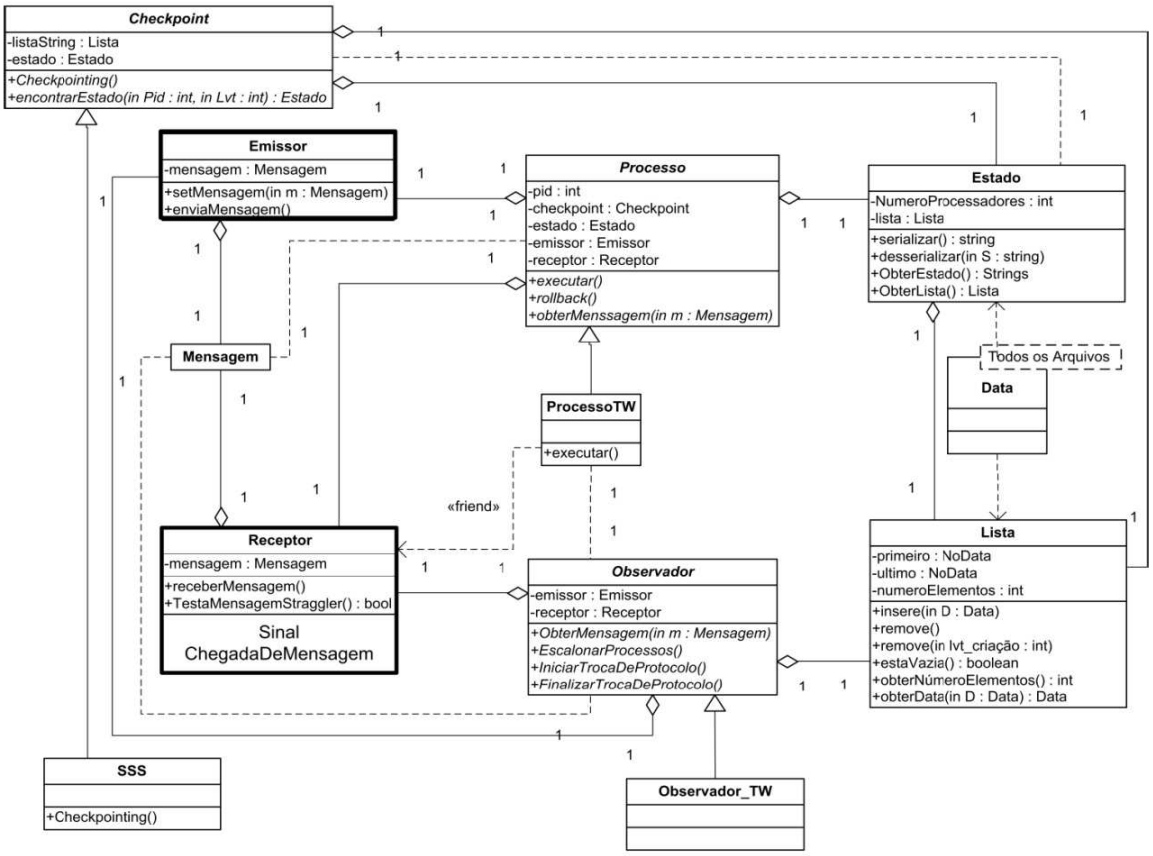

Figura 7: Diagrama de classes proposto para o *Time Warp* (CRUZ, 2009)

os *checkpoints* locais e pode ser redefinida utilizando o protocolo de marcação de *checkpoints* escolhido, dos quais, nesta especificação, é utilizado o *Fixed Dependency After Send* (FDAS). A implementação do processo observador, fundamental para a ação do *Rollback* Solidário, é determinada pela classe **Observador_SR** que possui métodos para coordenar as linhas de recuperação, além de escalonar processos e realizar troca de protocolos.

O protocolo *Time Warp* também foi acoplado ao *framework*. Ele apresenta uma estrutura semelhante a do protocolo *Rollback* Solidário, conforme pode ser visto no diagrama de classes da figura 7. Para este protocolo, a classe **Checkpoint** é implementada através do mecanismo *Sparse State Saving* (SSS). Outra diferença deste diagrama encontra-se na classe **Observador**. Neste caso, o obje-

tivo desta classe é permitir a implementação de mecanismos para a troca dinâmica de protocolos ou a migração de processos.

2.4.2.2 Adaptação do Framework Proposta por Azevedo (2012)

Azevedo (2012) propõe um *framework* que possui classes comuns ao de Cruz (2009), entretanto, o principal diferencial desta nova proposta está na implementação das classes e métodos relacionados à execução do modelo de simulação e ao protocolo de sincronização. Além disso, é utilizada a implementação do protocolo MPI (*Message Passing Interface*) em Java, através da biblioteca MPJ-Express (MPJ-EXPRESS, 2014), a fim de aproveitar a flexibilidade inerente à tecnologia Java, permitindo a execução de simulação distribuída em um ambiente multiplataforma. A seguir serão descritas as alterações mais relevantes em relação ao *framework* original.

A figura 8 apresenta a hierarquia de classes responsáveis por gerenciar os processos na simulação distribuída. Como pode-se observar, as nomenclaturas das classes e dos métodos foram traduzidas para o inglês. A classe **Process** proporciona uma abstração para um processo alocado na execução da simulação, correspondente a um processo real executando uma tarefa. Um objeto da classe **Process** também é utilizado para controlar a execução. Esta classe provê um método abstrato *execute()* onde a tarefa será implementada. Ainda contém o método protegido abstrato **getEnvironment()** permitindo que as subclasses acessem o atributo privado **environment** e utilizem os serviços fornecidos pela classe **Environment**.

A utilidade da classe **ControllerProcess** parte da premissa de que, independente do protocolo de sincronização, deverá existir sempre um processo controlador que, no mínimo, irá coordenar o início da simulação, através da operação **initializeSimulation()**, enviando os dados necessários para os processos efetivamente envolvidos na simulação. A classe **SimulationProcess** é utilizada para executar as tarefas de um processo da simulação. Todas as suas instâncias, referentes

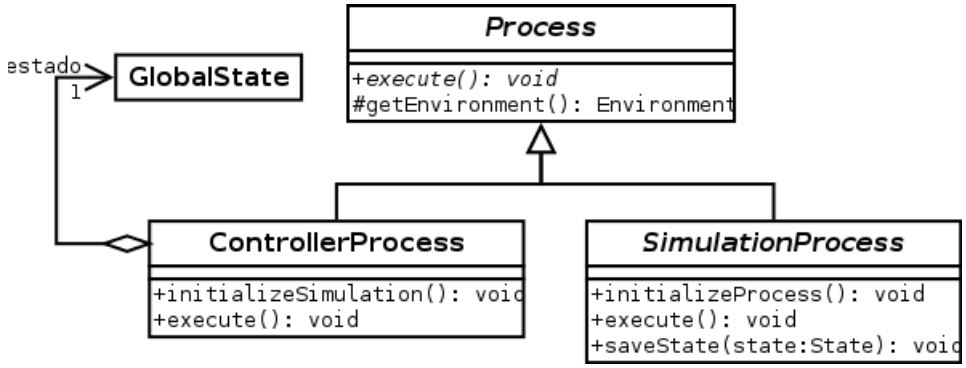

Figura 8: Classes responsáveis por gerenciar os processos envolvidos na simulação (AZEVEDO, 2012)

aos processos envolvidos na simulação, contribuem para que o modelo possa ser simulado.

Duas novas classes foram especificadas: **LocalState** e **GlobalState**, substituindo a classe original **EstadoGeral**. Elas servem para manter as informações de estados locais de cada processo e as informações de estado global da simulação. Cada processo da simulação contém uma instância de **SimulationProcess**, que envia, periodicamente, seus dados de estado local ao controlador (**ControllerProcess**). Por conseguinte, o controlador utiliza estes dados para compor o estado global da simulação. Desta forma, as classes de estado estão associadas às classes referentes aos processos, ou seja, **SimulationProcess** e **ControllerProcess**, diferente do versão de Cruz (2009), na qual a classe de estado tinha associação com à classe **Ambiente** (ou **Environment** na nova versão),

As classes envolvidas no procedimento de salvamento de estados e a classe responsável pelo protocolo de sincronização são exibidas na figura 9. Para realizar o salvamento de estados da simulação é utilizada a classe **StateSaver**. Ela é uma classe abstrata e, portanto, a implementação do algoritmo de salvamento fica a cargo de suas subclasses. Através da classe **SimulationProcess** é possível notificar a **StateSaver** quando eventos importantes provocarem o salvamento de estados.

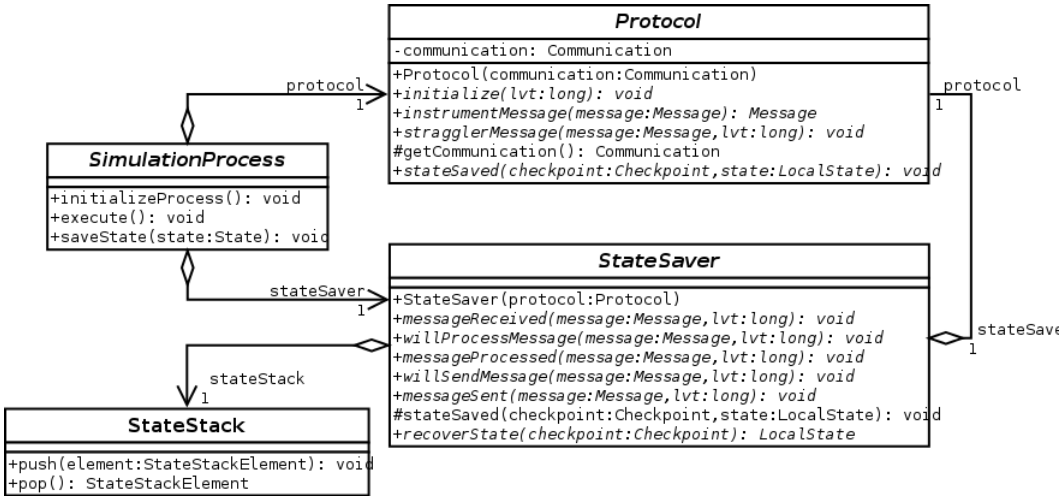

Figura 9: Classes envolvidas no salvamento de estados e responsáveis pelo protocolo de sincronização (AZEVEDO, 2012)

2.4.2.3 Melhorias no Framework Propostas por Faria (2016)

Conforme citado anteriormente, Faria (2016) propôs um método que determina quando é vantajoso realizar a troca dinâmica entre os dois protocolos de sincronização (*Time Warp* e *Rollback* Solidário). Fundamentado nesta técnica, ao final da simulação, os dados inerentes ao desempenho de cada processo do modelo simulado são gerados e enviados ao observador, para que sejam produzidos seus respectivos relatórios. A avaliação também considera o cálculo da média dos resultados para possibilitar um diagnóstico global, observando os dados de todos os processos. A partir da ponderação dos resultados, é possível determinar se existe a necessidade da troca de protocolos.

O recurso desenvolvido baseia-se no mesmo eixo do trabalho de Azevedo (2012), utilizando-se a implementação em Java e a biblioteca MPJ-Express para troca de mensagens entre processos. Neste caso, a especificação do *framework* proposta por Azevedo (2012) foi reutilizada para construção dos mecanismos apresentado por Faria (2016).

Neste contexto, o trabalho de Faria (2016) oferece melhorias em termos de

análise e otimização de uma simulação distribuída. As novas funcionalidades, adicionadas ao código base, possibilitaram a concepção e a implementação do método de comparação de protocolos otimistas. Todavia, o principal objetivo desta subseção, não é apresentar o método de análise de protocolos, propriamente dito, mas sim, as modificações na estrutura básica do *framework* em questão. Dentre as finalidades dos novos algoritmos, estão:

- particionamento da simulação em subintervalos: a subdivisão prévia da simulação em intervalos menores permite a análise da simulação em espaços particulares de tempo.
- cálculo de GVT: baseado no algoritmo de Mattern (1993), foi implementado o procedimento para obtenção do GVT, sendo adaptado, também, para o *Rollback* Solidário. Isto se deve à duas finalidades distintas: (1) controlar os tamanhos das listas, consequentemente otimizando o uso da memória (desta maneira, prevenindo situações de *over flow*), e (2) evitar que os dados referentes ao desempenho de um intervalo não interfira em seus intervalos adjacentes;
- geração de dados para análise de desempenho da simulação e estabelecimento da necessidade de realizar a troca de protocolos. O mecanismo implementado possibilita que a simulação possa continuar, nas mesmas condições em que parou no intervalo anterior, com um protocolo distinto.

Sobretudo, as novas implementações foram tratadas de forma a reduzir problemas relacionados ao crescimento do *heap* do Java, devido ao alto número de instanciações ocorridas a cada iteração da simulação. Além disso, outro recurso introduzido no *framework*, que também contribuiu no desempenho do mecanismo quanto à economia de memória, foi o padrão de projeto *Singleton*. Ele impede que vários objetos de uma classe sejam criados, limitando-as à apenas uma instância por classe durante toda a execução. Em vista disto, a memória será conservada, e assim, proporcionará melhor aproveitamento dos recursos computacionais.

2.5 Considerações Finais

Neste capítulo foram apresentados os conceitos fundamentais sobre Simulação Distribuída, com suas principais abordagens e protocolos utilizados para sincronização de processos. Além disso, foram descritas outras pesquisas acadêmicas, diretamente relacionadas a este trabalho, as quais estão voltadas ao desenvolvimento de aplicações para simulação que exploram os conceitos apresentados.

O conteúdo descrito neste capítulo é importante para a compreensão das pesquisas que vêm sendo realizadas na área de Simulação Distribuída pelo Grupo de Pesquisas em Engenharia de Sistemas e de Computação (GPESC), que estão diretamente relacionadas a este trabalho de mestrado. Principalmente, para compreender o comportamento do *framework* e os protocolos de sincronização acoplados nele. Sobretudo, esta revisão bibliográfica serve de apoio para uma melhor clareza dos mecanismos envolvidos na ferramenta proposta.

O motivo da utilização dos protocolos otimistas neste trabalho, em especial, o *Time Warp* e o *Rollback* Solidário, é devido ao fato deles já estarem implementados no *framework*. Entretanto, é possível fazer uso de outros protocolos, inclusive, daqueles relacionados à abordagem conservativa.

Pode-se encontrar, na literatura, diversas pesquisas recentes sobre protocolos de sincronização, tanto conservativos como otimistas. Para um estudo mais aprofundado sobre os protocolos de sincronização e também variações sobre os protocolos utilizados neste trabalho, pode-se consultar as seguintes referências:

- Fujimoto (2016): uma breve descrição desta área de estudo é apresentada e diversos tópicos de pesquisa são explorados, incluindo áreas como a simulação orientada a problemas para sistemas de larga escala; redes complexas; exploração de unidades de *hardware* e ambientes de computação em nuvem para processamento gráfico; simulação preditiva para a gerenciamento e otimização de sistemas; consumo de energia em plataformas móveis e *data centers*;

e composição de simulações heterogêneas.

- Kunz et al. (2016): um novo paradigma de modelagem dos eventos discretos é proposto, a fim de melhorar o desempenho e a eficiência de técnicas existentes relacionadas à paralelização. É introduzido o conceito de "eventos expandidos", que utilizam informações adicionais para apoiar algoritmos de sincronização que, por sua vez, seguem a abordagem conservativa.

- Fujimoto (2015): uma visão geral da pesquisa em simulação paralela e distribuída é apresentada, desde trabalhos tradicionais para tratamento de problemas, como a sincronização, até recentes trabalhos relacionados à execução de simulações de grande escala em plataformas de supercomputação. As direções das pesquisas futuras nesta área são exploradas.

- LaPre et al. (2015): uma nova abordagem para salvar o estado no *Time Warp*, armazenando apenas as mudanças relativas causadas por um evento.

- Harvey e Gentile (2015): proposta de uma nova abordagem alternativa aos protocolos tradicionais. A técnica consiste em sincronizar apenas as mudanças pertinentes às informações no modelo a cada avanço do tempo de simulação. Segundo os autores, serão transmitidos menos dados, reduzindo o tempo de execução da simulação.

- Bauer et al. (2015): uma nova técnica para controlar o otimismo em simulação distribuída de eventos discretos. É adequado para simular modelos em que os intervalos de tempo entre eventos sucessivos são altamente variáveis e não têm limites inferiores, melhorando o desempenho em relação a outras técnicas existentes que tentam prever a chegada de eventos por meio de estatísticas.

- Swenson, Ivey e Riley (2014): análise de desempenho sobre protocolos de sincronização conservativos utilizando um simulador de redes.

- Bragard, Ventresque e Murphy (2014): proposta de dois algoritmos para sincronização de processos em uma simulação distribuída. O primeiro algoritmo requer algumas propriedades topológicas da rede, enquanto o segundo algoritmo funciona sem qualquer exigência. São realizadas comparações com outras abordagens, destacando os benefícios de se usar um balanceamento dinâmico de carga para simulações distribuídas.

- Munck, Vanmechelen e Broeckhove (2014): avaliação de desempenho de um simulador distribuído, utilizando vários protocolos de sincronização conservativos, executado em um ambiente multiprocessador. Uma análise foi realizada sobre uma abordagem híbrida, combinando dois protocolos tradicionais, possibilitando o aumento da robustez e permitindo melhor desempenho da simulação.

- Barnes Jr. et al. (2013): estudo sobre simulação de grande escala utilizando o *Time Warp* com computação reversa.

- Alzraiee, Zayed e Moselhi (2012): proposta de uma metodologia para sincronização em modelos híbridos, que integram sistemas de eventos discretos e sistemas dinâmicos.

- Carothers e Perumalla (2010): estudo sobre o comportamento dos protocolos de sincronização em núcleos com alto poder de processamento.

- Kawabata et al. (2006): avaliação de desempenho do protocolo conservativo CMB (*Chandy-Misra-Bryant*)[1] a partir da perspectiva do tempo de execução. Os autores analisam o desempenho de cada processo lógico na simulação.

Além das contribuições dos trabalhos que foram discutidos neste capítulo, existem outros estudos desenvolvidos no GPESC, que, também, estão relacionados à

[1]O primeiro mecanismo de sincronização para simulação distribuída seguia a abordagem conservativa. Os primeiros algoritmos para este tipo de abordagem foram desenvolvidos, independentemente, por Chandy e Misra (1979) e Bryant (1977), que originaram o protocolo denominado CMB, em homenagem aos seus criadores.

área de Simulação Distribuída. O trabalho de mestrado de Junqueira (2012) apresenta um estudo detalhado sobre migração de processos em simulações distribuídas e propõe dois métodos para tratar esta migração. A ideia consiste em identificar um, eventual, desbalanceamento de carga entre os nós de um sistema distribuído e, por conseguinte, reescalonar os processos de maneira que haja um melhor equilíbrio do sistema.

Barbosa (2012) especificou as características de uma ferramenta paralela para simulação de eventos discretos. O autor discutiu os principais elementos que este tipo de projeto exige e destacou que a utilização dos mecanismos de simulação distribuída desenvolvidos no grupo, como o *framework* e os algoritmos de migração dos processos são meios que facilitarão o desenvolvimento de uma ferramenta para simulação distribuída. Em outro trabalho, Carvalho (2013) propôs uma nova métrica e um algoritmo de balanceamento que permite a avaliação das métricas em um ambiente de execução, com as mesmas características, como modelos de simulação e fatores de desbalanceamento de carga.

Vaz (2015) apresenta uma solução, que auxilia a analise, através de um simulador de computação em nuvem, de todo o histórico de envio e recebimento de mensagens, assim como os atrasos envolvidos em relação à rede. Com esta solução, é possível propor otimizações nos protocolos de simulação, melhorando o seu desempenho e confiabilidade durante a execução neste tipo de arquitetura computacional.

Neste contexto, a ferramenta proposta neste trabalho de mestrado foi desenvolvida a partir da conjuntura das pesquisas realizadas no GPESC, a fim de apresentar um arcabouço que abrange os mecanismos produzidos nos trabalhos anteriores. Sendo assim, esta ferramenta pode ser considerada um "produto tecnológico" produzido em consequência destes estudos.

O próximo capítulo apresenta um estudo sobre as ferramentas disponíveis para simulação de eventos discretos, com foco, em especial, naquelas que fornecem recursos para a Simulação Distribuída.

3 Ferramentas Para Simulação de Eventos Discretos

Neste capítulo será apresentado um estudo sobre as ferramentas para Simulação de Eventos Discretos (SED) de código aberto, que oferecem suporte à execução paralela, mostrando as principais características e critérios utilizados para avaliar este tipo de ferramenta. O objetivo deste estudo foi identificar requisitos para o desenvolvimento da ferramenta que foi desenvolvida neste trabalho de mestrado, inclusive, permitindo apresentar, posteriormente, as vantagens da ferramenta proposta em relação às já existentes. Além disso, são fornecidos aos pesquisadores informações sobre as ferramentas paralelas de SED e potencialidades para trabalhos futuros. Uma análise comparativa das ferramentas é apresentada, mencionando as contribuições, vantagens e suas restrições. Em complemento, é apresentado uma análise geral sobre as ferramentas para SED, fornecendo informações a respeito das características envolvidas neste tipo de aplicação.

Atualmente, vários ambientes e ferramentas de simulação são encontrados na literatura, sendo a maioria de uso comercial. Exemplos destas ferramentas são: ASDA (BRUSCHI et al., 2004), Simul8 (CHWIF; MEDINA, 2006), ExtendSim (KRAHL, 2013), ProModel (KHALILI; ZAHEDI, 2013), Simio (THIESING; PEGDEN, 2013), Ururau (RANGEL; CORDEIRO, 2015), SAS Simulation Studio (HUGHES et al., 2013), Arena (KELTON; SADOWSKI; STURROCK, 2014) entre outros *softwares* disponíveis, conforme uma pesquisa do Instituto de Pesquisa Operacional e Ciências da Administração - INFORMS (SWAIN, 2015).

A literatura apresenta alguns estudos e comparações sobre ferramentas para simulação de eventos discretos (SARKAR; HALIM, 2011; WEINGÄRTNER; LEHN; WEHRLE, 2009). Um recente trabalho mostrou uma revisão sobre *softwares* de simulação de eventos discretos para pesquisa operacional, focando apenas em ferramentas *Open Source* (DAGKAKIS; HEAVEY, 2015). Além disso, outros estudos sobre as abordagens de paralelização vem apresentando contribuições importantes para a evolução tecnológica no campo da Simulação, conforme apresentado no trabalho de Fujimoto (2016). No entanto, muitos desafios estão presentes nesta área científica, a qual, por sua vez, possui tendência de crescer ainda mais.

Tendo em vista que diversas pesquisas vêm sendo realizadas a fim de aumentar o conhecimento a respeito do avanço das tecnologias utilizadas na área de simulação, o eixo da análise discutida neste capítulo é o uso da simulação paralela. Neste contexto, algumas questões de pesquisa foram levantadas:

1. O processamento paralelo e distribuído vem sendo aplicado na Simulação de Eventos Discretos para resolver quais tipos de problemas?

2. Qual o nível de portabilidade destas ferramentas? Ou seja, quais plataformas ou Sistemas Operacionais são suportados para a execução?

3. Quais ferramentas são comerciais (proprietárias)? Quais apresentam versões gratuitas? Quais são de código aberto?

4. Quais ferramentas de código aberto são capazes de simular modelos grandes e complexos com desempenhos satisfatórios?

5. Qual abordagem de paralelização vem sendo utilizada nas ferramentas de simulação?

3.1 Metodologia para Análise das Ferramentas

A metodologia utilizada na análise das ferramentas de simulação foi baseada no trabalho de Dagkakis e Heavey (2015). Será explicado como foi realizada a busca na literatura acadêmica e na internet, para que fosse possível identificar as ferramentas de SED que oferecem recursos para execução distribuída ou paralela. Então, foram definidos critérios, segundo os quais as ferramentas foram comparadas.

A pesquisa foi restringida a determinadas características a fim de focar nas questões levantadas e nas ferramentas relacionadas à proposta deste trabalho de mestrado. O filtro imposto nesta análise considerou as seguintes definições:

- Neste trabalho foram consideradas apenas ferramentas para simulação de eventos discretos. Ferramentas que estão focadas em outras técnicas de simulação foram desconsideradas.
- O foco da análise são as ferramentas que oferecem recursos para simulação distribuída e paralela. Todavia, foram levantados, também, a lista daquelas que apresentam apenas execução sequencial.
- A coleta de material, inicialmente, considerou ferramentas que são voltadas a diversos domínios de aplicação específicos, sendo possível obter uma visão ampla sobre as ferramentas paralelas para simulação de eventos discretos. No entanto, a segunda fase da análise deu ênfase a ferramentas que abrangem um domínio de aplicação genérico.
- O filtro final corresponde ao direito de propriedade autoral das ferramentas, sendo escolhidas aquelas que apresentam código aberto.

O ponto de partida para a coleta das ferramentas foi a literatura acadêmica. As buscas foram feitas utilizando, como palavra-chave, uma combinação de termos: *('Tool' OR 'Software' OR 'Environment') AND ('Discrete Event' OR 'Discrete-Event') AND 'Simulation'*. Vale destacar que a pesquisa foi realizada nas principais

bases de dados bibliográficos, como: *Web of Science*, *SciVerse Scopus*, *ACM Digital Library* e *IEEE Explorer*. No entanto, esta revisão é mais ampla, considerando também as ferramentas que não estão na literatura acadêmica, sendo utilizado o *Google* como mecanismo de coleta.

3.1.1 Definições de Critérios de Avaliação

Existem diversos estudos, amplamente discutidos na literatura acadêmica, sobre avaliação e seleção de *software* (TEWOLDEBERHAN et al., 2002; JADHAV; SONAR, 2009). Da mesma forma, também há estudos focados na seleção de *softwares* de simulação (AYAĞ; SAMANLIOGLU; YÜCEKAYA, 2012; ALOMAIR; AHMAD; ALGHAMDI, 2015; DAGKAKIS; HEAVEY, 2015). As referências citadas contribuem para estabelecer um contexto teórico a respeito dos critérios de avaliação deste tipo de ferramenta, no entanto, não foi possível encontrar pesquisas que têm como foco a avaliação de ferramentas para simulação distribuída e paralela.

A seguir serão apresentados os critérios estabelecidos e as justificativas para escolha de cada um, apontando as inferências que estas medidas podem ter.

- *Modo de Execução*: pode ser categorizado em grupos que distinguem a maneira de como é realizado o processamento de uma simulação no sistema computacional: *Simulação Distribuída* (a execução de cada replicação é dividida em diversos processadores) ou *Simulação Sequencial* (é utilizado apenas um processador para executar cada replicação). Nesta pesquisa, foram analisadas ferramentas que suportam o modo de execução da primeira categoria citada.

- *Plataformas Suportadas*: trata-se da combinação de *hardwares* e sistemas operacionais que o *software* pode ser executado. A avaliação desta característica nas ferramentas de simulação permite identificar seus níveis de portabilidade, uma vez que algumas são multiplataformas. Além disso, é possível

fazer uma análise quantitativa sobre as plataformas suportadas por esses *softwares*.

- *Domínio de Aplicação*: estabelece os domínios de problemas que a ferramenta é capaz de representar. Conforme citado na seção de metodologia, este critério pretende coletar informações sobre ferramentas de simulação que podem ser utilizadas para domínios genéricos.

- *Custo e Propriedade*: o custo é um dos fatores que impactam na utilização do *software*. Muitos *softwares* comerciais apresentam recursos robustos e capazes de representar de forma intuitiva os sistemas, porém, o custo destes pode restringir seu público alvo. Por outro lado, existem soluções gratuitas, inclusive, algumas de código aberto (*Open Source*), possibilitando o acesso mais amplo e o desenvolvimento colaborativo, o que contribui para o desenvolvimento e inovação na área da Simulação. Será utilizado o termo "direito autoral" para representar os direitos de propriedade referentes às ferramentas. Sendo assim, os projetos das ferramentas comerciais possuem direitos autorais "proprietários" e, portanto, o código fonte não é disponibilizado, além da possibilidade de envolver custos para obtenção de licença do *software*. Por outro lado, existem as ferramentas com direitos autorais de "código aberto", que oferecem diversos tipos de licenças gratuitas para uso do projeto com o código fonte disponível (DAGKAKIS; HEAVEY, 2015).

3.2 Avaliação

Esta seção apresenta os resultados para os critérios descritos na seção anterior, exibindo uma análise das ferramentas com base nestas medidas. Na subseção 3.2.1 são apresentadas algumas considerações gerais sobre as características estatísticas dos resultados, a fim de proporcionar uma primeira visão genérica sobre o estado da arte em ferramentas paralelas para Simulação de Eventos Discretos. Para alcançar a análise específica proposta no início deste capítulo, a avaliação abrangeu duas

etapas que consistiram em: filtrar um subconjunto das ferramentas com base nos critérios estabelecidos e, por conseguinte, avaliar as ferramentas selecionadas.

3.2.1 Análise Geral

Os resultados completos são apresentados nas tabelas 1 e 2, onde são mostrados, para cada ferramenta, as informações referentes ao domínio de aplicação, às plataformas suportadas para execução e ao tipo de propriedade autoral. Foram identificadas no total 63 ferramentas de SED, sendo que 28 oferecem recursos para paralelização (tabela 1) e 35 sequenciais (tabela 2). Com base nestas respostas, foi possível obter algumas considerações gerais sobre as Ferramentas Paralelas para Simulação de Eventos Discretos.

Tabela 1: Lista das Ferramentas Paralelas de Simulação de Eventos Discretos

	Software	Domínio	Plataformas	Propriedade
1	AnyLogic	Genérico	Windows, Mac, Linux	Proprietária*
2	Arena	Genérico	Windows	Proprietária*
3	CSIM 20	Sistemas Computacionais	Windows, Linux	Proprietária**
4	Enterprise Dynamics	Genérico	Windows	Proprietária*
5	FlexSim	Genérico	Windows	Proprietária*
6	GoldSim	Genérico	Windows	Proprietária*
7	Micro Saint Sharp	Genérico	Windows	Proprietária**
8	ProModel	Genérico	Windows	Proprietária**
9	Proof Animation	Logística / Manufatura	Windows	Proprietária*
10	SAS Simulation Studio	Genérico	Windows	Proprietária**
11	Simio Design/Team	Genérico	Windows	Proprietária**
12	SIMPROCESS	Hierarquias/Genérico	Windows, Linux	Proprietária**
13	SIMUL8	Genérico	Microsoft Windows	Proprietária**
14	Tecnomatix Plant Simulation	Logística / Manufatura	Windows	Proprietária*
15	WITNESS	Genérico	Windows	Proprietária*
16	ASDA	Sistemas Computacionais	Linux	Open Source
17	CD+ +	Genérico	Linux	Open Source
18	JaamSim	Genérico	Windows, Mac, Linux	Open Source
19	J-Sim	Biomedicina	Windows, Mac, Linux	Open Source
20	MGSim	Sistemas Embarcados	Linux, FreeBSD, Solaris	Open Source
21	OMNeT + +	Genérico	—	Open Source
22	PARSEC	Simulação Paralela	Linux	Open Source
23	RePast	Genérico	Windows, Linux	Open Source
24	Root-Sim	Simulação Paralela	Linux	Open Source
25	SIMCAN	Redes de Comunicação	Linux	Open Source
26	SimGrid	Sistemas Embarcados	Windows, Linux, Mac	Open Source
27	SimKit	Genérico	Windows	Open Source
28	Spades/JAVA	Inteligência Artificial	Linux	Open Source

As ferramentas proprietárias podem estar em três grupos:

1. Possui versão gratuita (*).

2. Não possui versão gratuita, mas apresenta versão com custo reduzido (**).

3. Não possui versão gratuita (***).

Tabela 2: Lista das Ferramentas Sequenciais de Simulação de Eventos Discretos

	Software	Domínio	Propriedade
1	GPSS	Pesquisa Operacional	Proprietária**
2	BLUESSS simulation package	Genérico	Proprietária***
3	ExtendSim AT	Pesquisa Operacional	Proprietária**
4	GPSS/H	Genérico	Proprietária*
5	SLX	Genérico	Proprietária*
6	C+ + Sim	Genérico	Open Source
7	DESMO-J	Genérico	Open Source
8	Facsimile	Manufatura	Open Source
9	FreeSML	Genérico	Open Source
10	GeneSim	Gerador de Código	Open Source
11	Hase	Sistemas Computacionais	Open Source
12	Jades	Genérico	Open Source
13	JAPROSIM	Genérico	Open Source
14	JavaSim	Genérico	Open Source
15	JGPSS	Educacional	Open Source
16	JiST	Redes de Comunicação	Open Source
17	JSL	Educacional	Open Source
18	libFAUDES	Manufatura	Open Source
19	NS-3	Redes de Comunicação	Open Source
20	OOSimL	Educacional	Open Source
21	OSSim	Educacional	Open Source
22	OverSim	Redes de Comunicação	Open Source
23	PowerDEVS	Genérico	Open Source
24	PySimulator	Genérico	Open Source
25	ScipySim	Genérico	Open Source
26	SharpSim	Genérico	Open Source
27	SIM.JS	Genérico	Open Source
28	SimJava	Genérico	Open Source
29	SimPy	Genérico	Open Source
30	SystemC	Sistemas Embarcados	Open Source
31	TerraME	Sistemas Terrestres	Open Source
32	Turtuga	Genérico	Open Source
33	URURAU	Cadeias de Suprimentos	Open Source
34	XGDESK	Genérico	Open Source
35	YetiSim	Genérico	Open Source

O gráfico, ilustrado na figura 10, apresenta a relação dos direitos autorais e custos de todas as 63 ferramentas coletadas. Conforme já discutido, os direitos autorais de *software* podem estar em duas categorias distintas: proprietários ou código aberto. Neste gráfico, os dados referentes às ferramentas paralelas e sequenciais são mostrados separadamente. Sendo assim, é possível analisar, em termos

quantitativos, o tipo de direitos autorais dos dois grupos de ferramentas. As ferramentas de código aberto compõem a maior parte do conjunto com quase 70% do total. Por sua vez, as ferramentas proprietárias são caracterizadas por serem comerciais, no entanto, a grande maioria delas apresenta versões para estudante. Como pode-se observar, metade das ferramentas comerciais disponibiliza versões gratuitas e quase todo o restante oferece versões com descontos.

Em relação ao conjunto de ferramentas paralelas, a proporção entre as que são de código aberto e as proprietárias é praticamente igual, contendo 13 e 15, respectivamente. Dentre às paralelas comerciais, 7 possuem versões com desconto e 8 disponibilizam opções gratuitas. Entre as ferramentas que executam apenas de modo sequencial, a quantidade que apresentam código aberto é bem superior ao número de ferramentas comerciais. Neste caso, apenas 5 ferramentas possuem direito autoral proprietário, enquanto 30 são de código aberto.

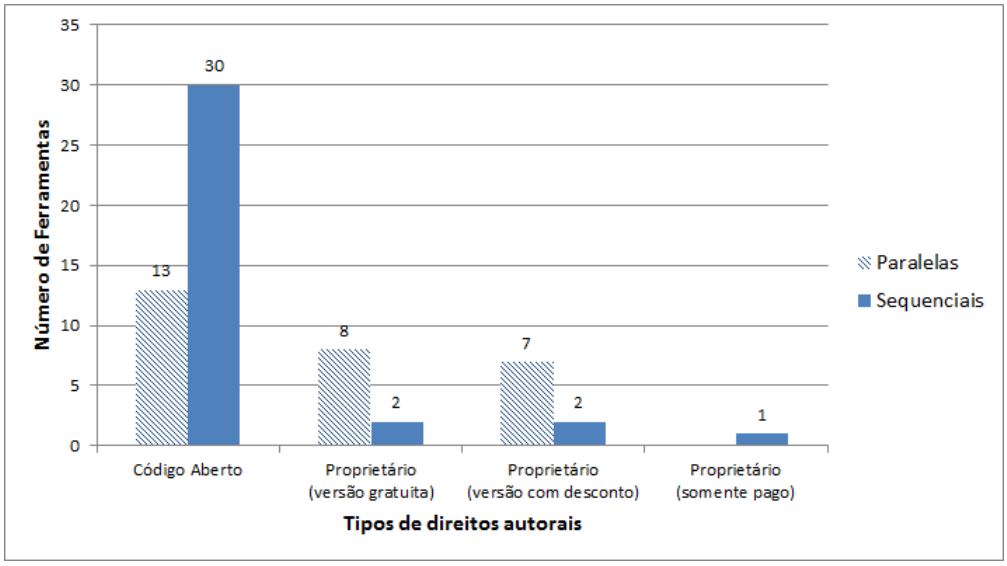

Figura 10: Direitos autorais das ferramentas

Os resultados apresentados nos parágrafos anteriores mostraram alguns dados a respeito do conjunto completo coletado nesta pesquisa, a fim de expor uma visão geral sobre as ferramentas de SED. O próximo gráfico apresentará dados sobre as plataformas suportadas, considerando apenas as ferramentas paralelas.

Desta forma, os resultados sobre as plataformas suportadas pelas ferramentas paralelas são retratados na figura 11. Como pode-se notar, as plataformas com Windows e Linux são as mais populares entre as ferramentas paralelas de SED. Ferramentas para o ambiente Mac ocupam o terceiro lugar desta classificação. Por fim, foram identificadas duas outras plataformas que podem ser utilizadas na paralelização da SED: FreeBSD e Solaris. Elas são suportadas, apenas, pela ferramenta MGSim (LANKAMP et al., 2013). Vale destacar que os valores apresentados neste gráfico não representam o somatório das ferramentas paralelas, uma vez que algumas delas estão disponíveis em mais de uma plataforma, por exemplo, JaamSim (KING; HARRISON, 2013) que pode ser executada em plataformas baseadas em Windows, Linux e Mac.

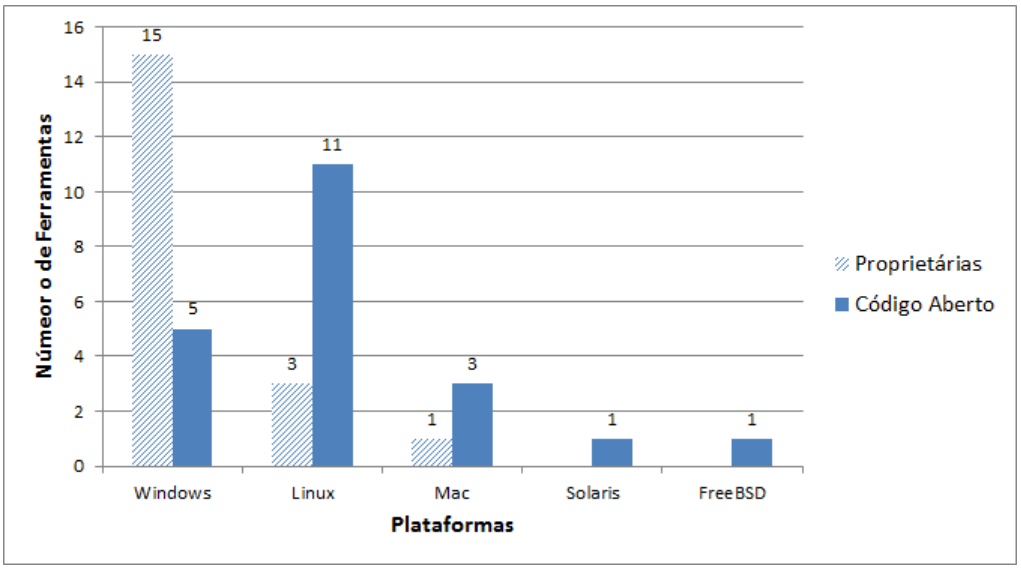

Figura 11: Plataformas suportadas pelas ferramentas paralelas

3.2.2 Primeira Fase: Filtragem

Nesta avaliação, foram consideradas apenas algumas ferramentas do conjunto coletado, devido a várias razões. Conforme estabelecido na metodologia, esta análise compreendeu uma parcela específica das ferramentas de simulação. Além disso,

foi necessário dar atenção a ferramentas que apresentam suporte na utilização do processamento paralelo através de fóruns, tutoriais ou artigos recentes mostrando sua relevância, uma vez que não é possível quantificar todas as medidas relevantes.

Com base nos dados coletados, foi possível estabelecer a lista das ferramentas resultantes que apresentam estas características. Com esta filtragem, obteve-se as quatro ferramentas listadas a seguir:

- CD++: é baseada em um formalismo conhecido como DEVS (*Discrete Event System Specification*) (GLINSKY; WAINER, 2006).
- JaamSim: é uma das poucas ferramentas de código aberto que não foi originada pela pesquisa acadêmica, mas por meio de recursos empresariais (KING; HARRISON, 2013).
- OMNeT++: é um *framework* para SED, que possui diversas publicações desde 1997 e tornou-se uma das mais populares ferramentas de simulação de código aberto (VARGA, 2013).
- SimKit: um pacote que foi projetado para o desenvolvimento de programas de SED e tem sido usado para criar modelos em diversas áreas (BOZOGLAN; GüNAL, 2009).

3.2.3 Segunda Fase: Análise das Ferramentas Selecionadas

Nesta subseção, as ferramentas que passaram pela primeira fase de avaliação serão analisadas de forma mais detalhada. Como já discutido, nem todos critérios são quantitativos e, em muitos casos, uma avaliação específica é necessária para obter uma melhor compreensão.

A ferramenta CD++ consiste em um conjunto de recursos para modelagem e simulação, construídos na linguagem C++. Assim como qualquer ferramenta de SED com processamento distribuído, CD++ utiliza mecanismos para sincronização, de modo a garantir ordem cronológica de ocorrência dos eventos, uma vez que

os processos lógicos da simulação serão escalonados em processadores diferentes. Este tema vem sendo, amplamente, discutido na comunidade acadêmica durante décadas e, nos últimos anos, diversas propostas de algoritmos de sincronização foram apresentadas (HARVEY; GENTILE, 2015; FUJIMOTO, 2016). Esta ferramenta é baseada no formalismo DEVS, que permite a especificação modular de modelos usando uma abordagem hierárquica (GLINSKY; WAINER, 2006). Na literatura, existem pesquisas recentes que abordam este formalismo de especificação, as quais utilizam CD++ como referência. A ferramenta foi modificada para implementar DEVS paralelas e, assim, ser capaz de executar modelos paralelos eficientemente em um ambiente distribuído. Uma recente versão, denominada "CD++ Builder", apresenta outros recursos gráficos para modelagem, que consiste em um *plugin* para o ambiente de desenvolvimento Eclipse. A figura 12 mostra as características deste ambiente (BONAVENTURA; WAINER; CASTRO, 2013).

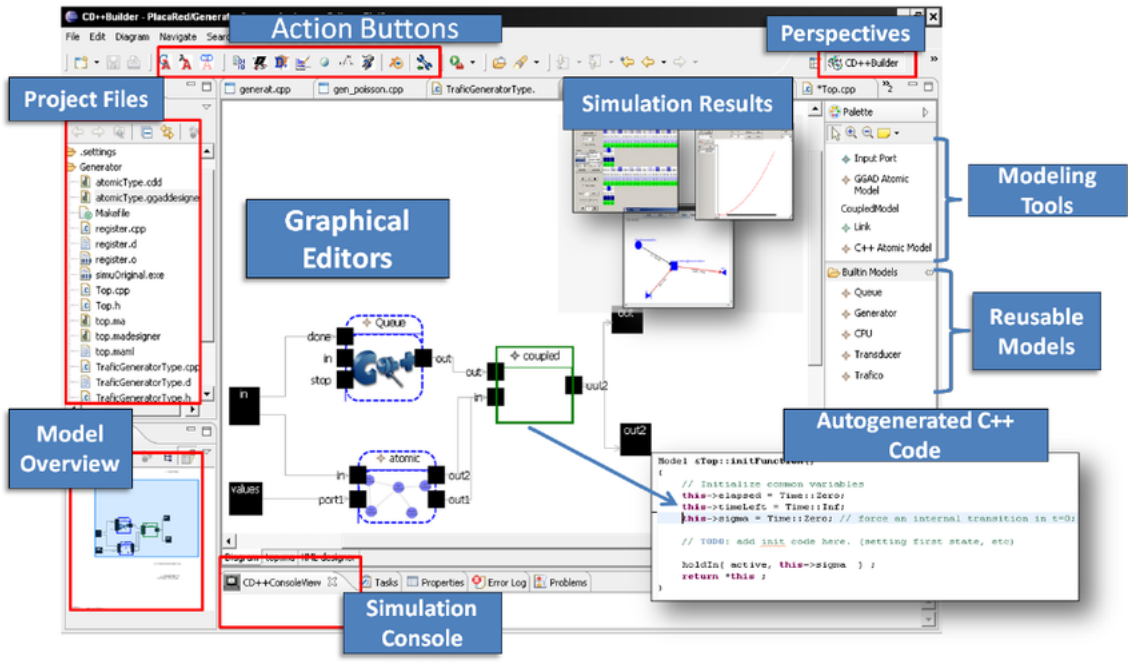

Figura 12: Características do ambiente CD++ Builder (BONAVENTURA; WAINER; CASTRO, 2013)

De acordo com Glinsky e Wainer (2006), CD++ utiliza como *middleware* o

kernel Warped, que fornece um mecanismo para a execução de simulação paralela. A ferramenta CD++ foi desenvolvida, inicialmente, para suportar dois núcleos diferentes: um que implementa o protocolo de sincronização *Time Warp* e outro que não fornece sincronização para ser utilizado em simulações sequenciais.

A ferramenta JaamSim é escrita na linguagem de programação Java. A principal característica que faz com que JaamSim seja diferente de outros *softwares* de simulação é a possibilidade do desenvolvimento de paletas personalizadas para objetos de alto nível em novas aplicações (KING; HARRISON, 2013). Segundo os autores, o termo "alto nível" refere-se a objetos significativos para uma determinada classe de modelo. Para um simulador de tráfego, por exemplo, os objetos de alto nível seriam os diversos tipos de veículos, estradas e cruzamentos. Também inclui uma GUI (*Graphical User Interface*) que é comparável àquelas fornecidas pelas ferramentas comerciais de simulação, como ilustrado na figura 13.

JaamSim foi desenvolvida a partir do *software* "*Transportation Logistics Simulator*" (TLS). Os autores disponibilizam vídeos no *Youtube* para demonstrar as capacidades da ferramenta e também mostrar exemplos do TLS. Novas paletas de objetos vêm sendo desenvolvidas e adicionadas à ferramenta para apoiar os projetos de modelagem. O manual do usuário da ferramenta JaamSim contém informações atualizadas sobre as GUIs e sobre novos recursos que são adicionados. Embora esta ferramenta tenha surgido de iniciativa empresarial, com base no *software* proprietário TLS, o código fonte é aberto.

OMNeT++ também suporta execução paralela de simulações de modelos complexos. Embora esta ferramenta esteja voltada a domínios como redes de comunicação, seus autores afirmam que sua arquitetura é genérica (VARGA, 2013). Ela fornece recursos avançados, para desenvolvedores e usuários, sendo implementada em C++. Ela oferece um Ambiente de Desenvolvimento Integrado (em inglês, *Integrated Development Environment* - IDE) que é baseado no Eclipse. O projeto oferece muitos tutoriais e documentos para suporte, além de apresentar uma lista de discussão utilizada, ativamente, pela comunidade (DAGKAKIS; HEAVEY, 2015).

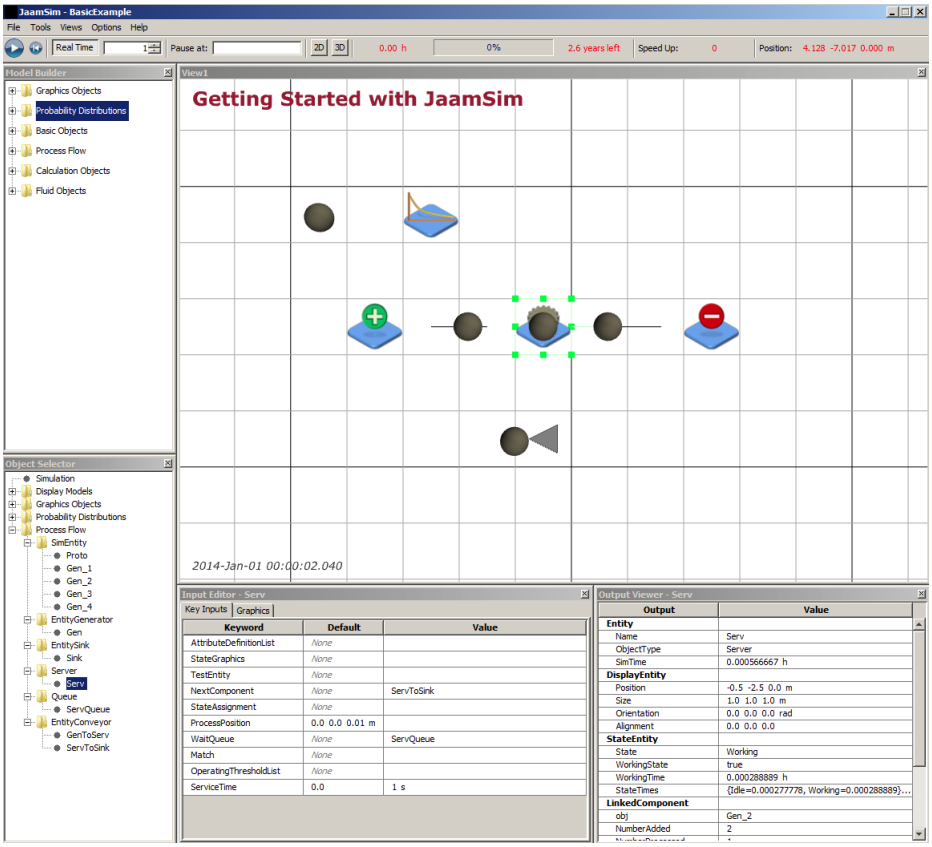

Figura 13: Interface gráfica da ferramenta JaamSim (KING; HARRISON, 2016)

O *framework* OMNeT ++ considera a utilização de métodos de sincronização conservativos e estatísticos na paralelização e possui uma interface de programação para enviar e receber mensagens entre as partições de modelo. A implementação desta interface permite o uso de uma biblioteca para troca de mensagens arbitrária. Desta forma, esta ferramenta oferece suporte para MPI e também para PVM (*Parallel Virtual Machine*). A troca de mensagens é transparente ao usuário mesmo entre diferentes arquiteturas computacionais. A figura 14 ilustra a IDE do OMNet++.

SimKit é um pacote que foi projetado para o desenvolvimento de programas de SED (BOZOGLAN; GüNAL, 2009). Possui implementações nas linguagens C++ e Java. A primeira proposta desta ferramenta foi apresentada em 1983, mas o seu

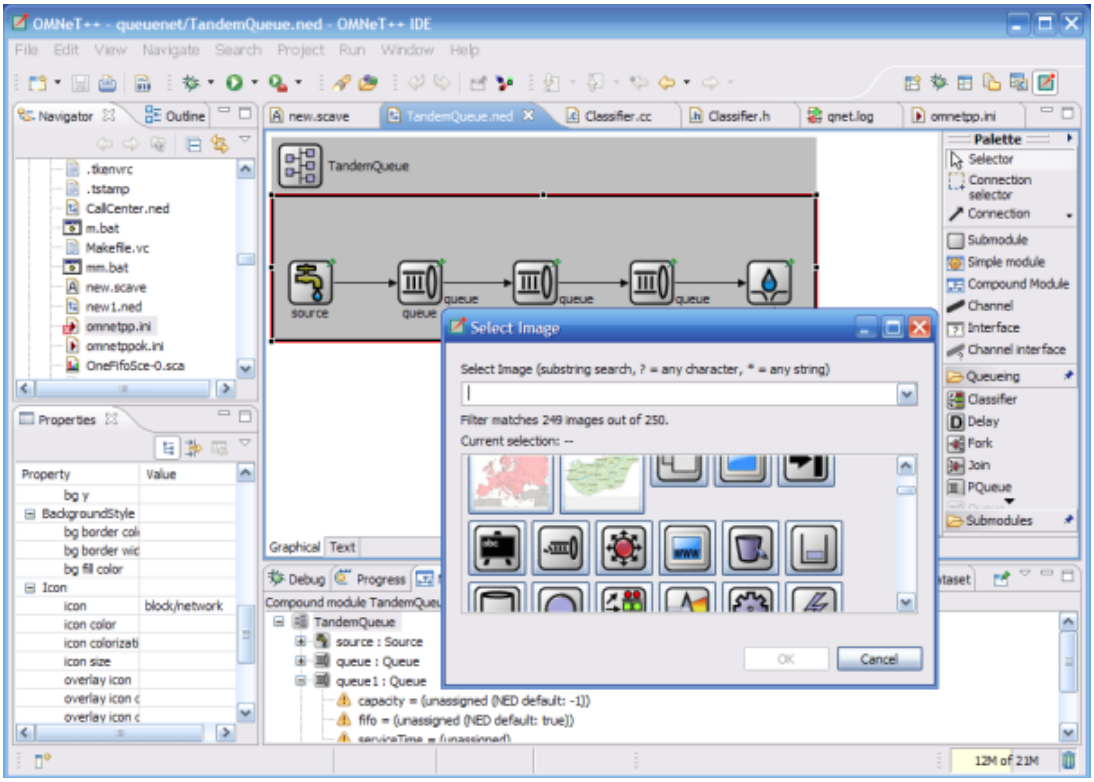

Figura 14: Ambiente de Desenvolvimento Integrado do OMNet++

desenvolvimento evoluiu ao longo dos anos. Simkit tem sido usado para criar modelos em diversas áreas, incluindo logística e apoio operacional. Várias restrições são impostas nesta ferramenta para permitir execução sequencial e paralela eficiente (BUSS, 2004). Embora existam poucas referências recentes na literatura sobre esta ferramenta, ela foi considerada nesta pesquisa por apresentar as características fundamentais relacionadas à paralelização da simulação, baseadas no protocolo *Time Warp*. A informações sobre a arquitetura básica destes mecanismos são descritas em Gomes et al. (1995).

3.3 Discussão

Esta seção apresenta uma discussão baseada nos resultados da análise, de modo a tratar as questões de pesquisa levantadas no início deste capítulo.

Pôde-se constatar que a área de Simulação Paralela e Distribuída tem sido, amplamente, discutida na literatura para resolver diversos tipos de problemas. Neste contexto, as ferramentas de SED vêm aprimorando mecanismos para suportar a execução paralela da simulação. A utilização deste tipo de solução é vantajosa quando os modelos são complexos e exigem o tratamento de uma grande quantidade de dados, pois o tempo de execução em um ambiente sequencial é inviável. A paralelização da SED é aplicada, também, em ferramentas voltadas para outros domínios de aplicação como sistemas computacionais, redes de comunicação, inteligência artificial e sistemas embarcados.

Em relação à portabilidade das ferramentas paralelas de SED, identificou-se que as comerciais são, em sua maioria, desenvolvidas para plataformas Windows. Por outro lado, a maior parte das ferramentas de código aberto possuem suporte para Linux. O sentido desta análise é avaliar o nível de portabilidade deste tipo de ferramenta, uma vez que a capacidade de um *software* ser compilado ou executado em diferentes arquiteturas, seja de *hardware* ou de *software*, é um fator relevante para os usuários.

A partir das características das ferramentas filtradas, foi possível identificar alguns pontos relevantes sobre este tipo de aplicação. As quatro ferramentas descritas foram implementadas utilizando o paradigma de programação orientado a objetos e foram baseadas nas linguagens C++ ou Java. Este é um tradicional paradigma adotado por aplicações de SED. Dagkakis e Heavey (2015) explicam que isto é devido ao fato da linguagem de programação SIMULA, projetada para apoiar a simulação de eventos discretos, ter sido a primeira que introduziu o conceito de classes. Existem algumas vantagens da utilização destas linguagens, em especial, a linguagem Java, que é caracterizada por sua independência de plataforma. Isso significa que o mesmo programa compilado pode ser executado em diferentes sistemas operacionais. Portanto, as ferramentas JaamSim e SimKit herdam esta característica da linguagem Java, e assim, há possibilidades de executar a simulação em um *cluster* ou rede de computadores com diferentes plataformas, no caso

de simulações distribuídas e paralelas. Este fato, contribui para o desenvolvimento de aplicações voltadas à Computação em Nuvem (ZEHE et al., 2015).

A disponibilização do código fonte e a existência de manuais com a documentação sobre a implementação das ferramentas favorecem estudos que envolvem os mecanismos de computação paralela como, por exemplo, protocolos para sincronização. Outra vantagem é a possibilidade de explorar o projeto destas ferramentas para realizar validações de novas técnicas relacionadas ao desempenho computacional como, por exemplo, algoritmos de balanceamento de carga e migração de processos. Além disso, é possível identificar as abordagens de paralelização utilizadas por estas ferramentas, que referem-se à maneira como será realizada a execução das replicações da simulação. As quatro ferramentas descritas utilizam a mesma abordagem de paralelização, porém, com diferentes tipos de protocolos para sincronização. CD++, JaamSim e SimKit implementam protocolos otimistas e OMNet++ é baseada em sincronização conservativa.

Por fim, diversos tópicos de pesquisas podem ser explorados, incluindo áreas como a simulação de sistemas de larga escala; redes complexas; exploração de unidades de *hardware* e ambientes de computação em nuvem para processamento gráfico; simulação preditiva para a gerenciamento e otimização de sistemas e consumo de energia em *data centers* (FUJIMOTO, 2016).

3.4 Considerações Finais

O estudo apresentado neste capítulo pode ser visto como um ponto de partida para novas pesquisas relacionadas à paralelização de ferramentas de SED e serve como um guia para utilização de ferramentas de código aberto. Além disso, pode ser utilizado no intuito de avaliar as potenciais ferramentas paralelas para Simulação de Eventos Discretos.

As ferramentas comerciais listadas nesta pesquisa apresentam versões para estudante, algumas com desconto e outras gratuitas. Embora elas ofereçam grande

suporte ao usuário de simulação e recursos robustos, o fato delas serem de código fechado, impossibilita obter respostas sobre o funcionamento dos mecanismos utilizados para executar simulações paralelas e distribuídas. Por sua vez, o desenvolvimento de ferramentas de código aberto que tratam a paralelização da SED têm-se mostrado um importante meio para a evolução deste tipo de mecanismo.

Em virtude disto, a ferramenta desenvolvida neste trabalho de mestrado será disponibilizada com o código aberto, de maneira a contribuir com a comunidade científica. Além disso, ela apresenta outras características, também existentes nas ferramentas destacadas neste capítulo, como por exemplo, a utilização de protocolos otimistas. Entretanto, foi possível observar que nenhuma das ferramenta analisadas permite a troca de protocolos de sincronização. Todas possuem implementações estáticas em relação aos métodos de sincronismo, sendo assim, não possuem flexibilidade para mudar a implementação de certos recursos inerentes ao protocolo. A possibilidade de alteração de protocolos de sincronização é relevante, pois utilizando protocolos adequados para determinados modelos ou áreas de aplicação é possível melhorar o desempenho da simulação.

Desta forma, a ferramenta que foi desenvolvida neste trabalho de mestrado foi projetada de modo a oferecer flexibilidade, permitindo a adição de outras implementações de protocolos, a fim de obter melhorias relacionadas ao desempenho da paralelização. Ela pode ser utilizada, também, para testes e validação de protocolos de sincronização, sendo esta, uma característica relevante para a área de Simulação Paralela e Distribuída. O próximo capítulo contém a descrição da ferramenta proposta.

4 Desenvolvimento da Ferramenta

Este capítulo apresenta a especificação da ferramenta proposta neste trabalho de mestrado, denominada "CronoSim". Trata-se de um *software* para simulação de eventos discretos, que pode ser executado de forma distribuída, ou seja, a simulação ocorrerá em uma plataforma computacional distribuída, conforme discussão realizada no capítulo 2. A principal característica, que difere o CronoSim de outras ferramentas de simulação disponíveis, é a capacidade de configurar ou selecionar protocolos de sincronismo da simulação e a infraestrutura computacional, possibilitando ainda, o uso por pesquisadores, para validação e testes de outros protocolos de sincronização.

Neste sentido, o projeto desta ferramenta foi baseado no propósito de criar um ambiente que, além de prover meios para melhorar o desempenho da simulação, auxilie usuários iniciantes em ambientes computacionais distribuídos na utilização, de uma maneira eficiente, dos recursos que este tipo de plataforma pode oferecer. Além disso, o monitoramento, associado a técnicas de escalonamento, permite realizar mapeamento dinâmico de processos, tornando vantajosa a simulação de modelos complexos.

Um fator importante no desenvolvimento deste trabalho foi a utilização do *framework* de Cruz (2009), apresentado no capítulo 2, que possibilitou a especificação de um ambiente para simulação de eventos discretos considerando detalhes já implementados. Apesar de existirem trabalhos que envolvam o projeto do *framework* em questão, não foi atribuído um nome a ele. Sendo assim, nesta dissertação será

utilizada a denotação "**CronoSim Framework**", a fim de referenciá-lo.

Este capítulo apresentará a especificação da ferramenta, mostrando também como o *framework* foi utilizado. As seções seguintes apresentam as principais características do projeto utilizando a UML (*Unified Modeling Language*) para especificação.

4.1 Tecnologias Utilizadas

A ferramenta desenvolvida foi implementada utilizando a linguagem Java e diversas bibliotecas, tanto nativas quanto externas, são exploradas. Entretanto, vale destacar o uso do padrão de comunicação MPI (*Message Passing Interface*) através da biblioteca "MPJ-Express" (MPJ-EXPRESS, 2014) e, também, utilização das biblioteca "JGraphx" (JGRAPH, 2006) e "iText" (ITEXT, 2016). Todas são *open source*, sob as licenças MIT (criada pelo *Massachusetts Institute of Technology*), BDS (*Berkeley Software Distribution*) e AGPL (*Affero General Public License*), respectivamnte (FSF, 2015).

Os motivos da escolha do Java são as principais características desta linguagem. Ela é uma das mais poderosas ferramentas da atualidade para o desenvolvimento de vários tipos de aplicações. Isto porque ela é segura, estável, oferece uma extensa biblioteca de programação e é uma linguagem orientada a objetos. Além disso, a linguagem Java é caracterizada por sua independência de plataforma. Isso significa que o mesmo programa compilado pode ser executado em qualquer sistema operacional. Portanto, uma outra vantagem é a possibilidade de programas serem executados em uma rede de computadores com diferentes sistemas operacionais (DEITEL; DEITEL, 2014).

A biblioteca MPJ-Express é responsável pela comunicação entre os processos durante a execução de um programa paralelo. Ela é implementada em Java, o que facilita a sua instalação e configuração em relação a outras implementações MPI nativas de sistemas operacionais Linux (AZEVEDO, 2012). Sendo assim, não

há necessidade de qualquer implementação MPI nativa para sua execução. Esta é uma das vantagens, pois permite a execução de uma aplicação distribuída em uma infraestrutura com plataformas heterogêneas.

A biblioteca JGraphx está relacionada à interface gráfica, possibilitando o desenvolvimento de recursos visuais, tais como manipulação de diagramas, visualização e interação com vértices e arestas de grafos. Ao importar esta biblioteca, é possível explorar diversas funcionalidades que permitem desenhar, interagir e associar um contexto utilizando a representação de diagramas. Portanto, com o JGraphx foi possível implementar os recursos que permitem a elaboração visual de modelos para a simulação.

A ferramenta CronoSim gera relatórios inerentes aos resultados da simulação, de modo que o usuário possa exportar em um arquivo no formato PDF (*Portable Document Format*). Para tanto, é utilizada a biblioteca iText com recursos que permitem a adaptação, inspeção e manutenção de documentos em PDF. Esta biblioteca possibilita gerar documentos a partir de arquivos XML (*eXtensible Markup Language*), banco de dados ou através de métodos específicos em Java (ITEXT, 2016).

A especificação do projeto da ferramenta foi realizada utilizando a UML (*Unified Modeling Language*) a fim de representar a arquitetura e os princípios de funcionamento. Para melhor compreensão do projeto, as próximas seções descrevem estes itens.

4.2 Arquitetura da Ferramenta

Uma característica importante da linguagem Java é possibilidade de separar as classes em pacotes, de acordo com suas pertinências. Isto facilita a manutenção dos componentes de um projeto e eventuais reutilizações de código, permitindo, inclusive, que projetos importem classes de outros pacotes (DEITEL; DEITEL, 2014).

CronoSim apresenta uma arquitetura composta por diversos pacotes com funcionalidades específicas. Além disto, outros componentes compõem o projeto, tais como, o **CronoSim Framework** e as bibliotecas, conforme foi explicado na seção anterior. Esta organização separa os níveis de abstração considerando a programação orientada a objetos. A relação entre os componentes envolvidos na implementação da ferramenta é ilustrada na figura 15.

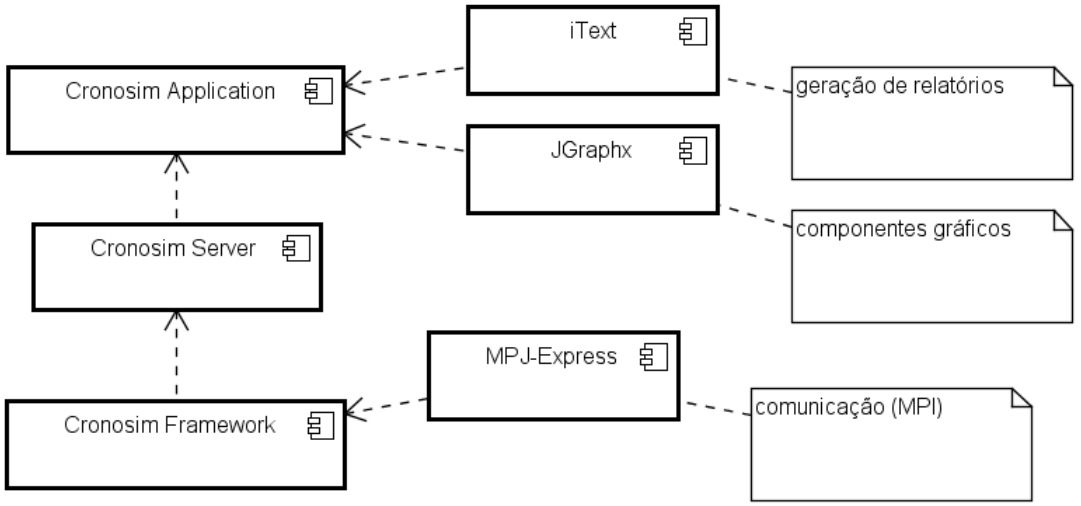

Figura 15: Diagrama de Componentes do CronoSim

A aplicação principal (**CronoSim Application**) abrange a maior parte das classes que foram desenvolvidas neste trabalho. Nela são concentradas todas as funcionalidades relacionadas à interface entre o usuário e o programa de simulação distribuída. Desta forma, este componente contém as classes responsáveis por controlar a Interface Gráfica do Usuário (do inglês, *Graphical User Interface* - GUI). Por meio da GUI, o usuário é capaz de interagir com o sistema através de elementos gráficos como ícones e outros recursos visuais.

Dentre as atribuições estabelecidas neste componente da arquitetura, destacam-se:

- Inicializar a ferramenta e possibilitar a elaboração e manipulação de modelos,

através dos recursos oferecidos pela interface gráfica.

- Permitir a definição das propriedades de uma simulação. Isto significa que o usuário poderá estabelecer os parâmetros correspondentes ao sistema a ser simulado, tais como o tempo lógico total da simulação, o número de replicações, unidades de tempo, entre outros. O funcionamento disso será detalhado nas próximas seções.

- Realizar a simulação dos modelos criados, de modo a propiciar ao usuário a escolha da forma de execução: simulação local (considerando apenas a máquina do usuário) ou simulação distribuída em uma rede de computadores.

- Fornecer recursos para que seja possível analisar os resultados da simulação após o término. Neste caso, os relatórios podem ser gerados e visualizados pelo usuário.

Para atender a estas responsabilidades, a aplicação faz uso dos componentes JGraphx e iText, referentes ao editor gráfico utilizado durante a modelagem e à geração de relatório, respectivamente. Além das atribuições citadas, o componente **CronoSim Application** envolve diversos métodos que controlam outros aspectos importantes da ferramenta, como:

- Conexão com o sistema distribuído: a aplicação se comunica com um *cluster* por meio de *sockets*. O componente **CronoSim Server** é responsável por receber as solicitações da aplicação principal e tratar no lado do servidor. Com esta arquitetura, o controle sobre a execução é realizada no próprio *cluster*.

- Controle durante a simulação: quando o usuário solicita o início da simulação, alguns métodos são invocados, a fim de gerenciar a sua execução. Inicialmente, uma *thread* é disparada, sendo responsável por comunicar com o servidor no *cluster* onde será realizado o processamento distribuído. Depois de estabelecer comunicação e a simulação tiver sido iniciada, esta *thread*

recebe do servidor informações sobre o *status* da simulação, enquanto fica aguardando o seu término. Portanto, o usuário poderá obter informações atualizadas sobre a simulação em tempo de execução.

O componente **CronoSim Server** também representa parte significativa do projeto, contendo classes que foram desenvolvidas para estabelecer a conexão entre a aplicação principal e uma rede de computadores. Ele está associado ao *cluster* onde será realizada à execução da simulação. O mecanismo implementado neste pacote da ferramenta funciona como um serviço em uma máquina *front-end* no lado do servidor, sendo responsável pela comunicação entre o sistema distribuído e a aplicação principal, conforme foi explicado anteriormente. O seu princípio de funcionamento baseia-se em *server socket* iniciado por uma *thread*, permitindo, assim, a conexão de múltiplos clientes simultaneamente. Em outras palavras, o servidor no qual a simulação será realizada possui um serviço que está sempre a espera de requisições de novas simulações. A figura 16 ilustra como é realizada a comunicação entre a aplicação do usuário e o *cluster*.

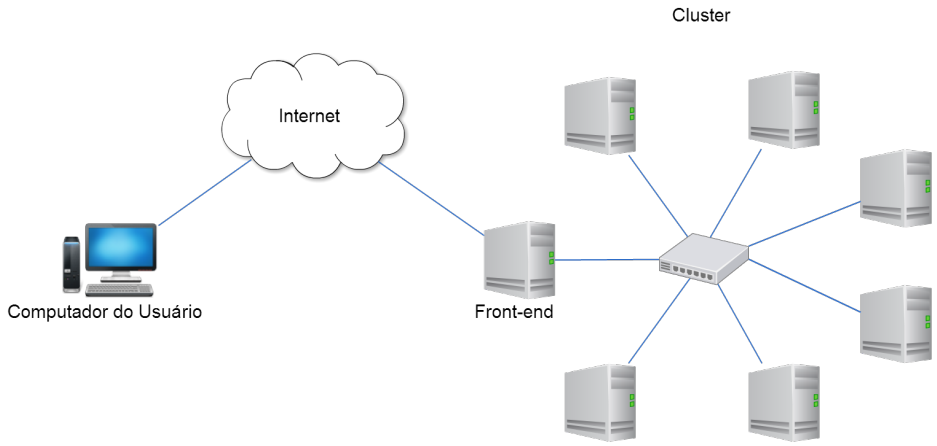

Figura 16: Arquitetura de comunicação entre a aplicação principal e o *cluster*

O computador do usuário, onde é executada a aplicação principal, pode conectar-se a um *cluster* disponível em rede local ou em outra rede de computadores externa através da internet (figura 16). Dessa forma, pode-se tirar proveito de uma infraestrutura robusta para executar a simulação, como o uso de Computação em Nuvem

e, por conseguinte, evitar altos investimentos na compra de *hardwares* e *softwares*. Para tanto, o serviço implementado pelo **CronoSim Server** deve estar ativo na máquina *front-end* do *cluster*. Além disso, os recursos do **CronoSim Framework** serão utilizados em cada computador empregado na simulação.

Após a modelagem e especificação das propriedades da simulação na camada de aplicação e o início do procedimento de execução por meio do **CronoSim Server**, a ferramenta faz uso dos recursos oferecidos pelo **CronoSim Framework** para gerenciar o processamento distribuído. Como foi mostrado na figura 15, o uso do MPJ-Express permite a troca de mensagens entre as máquinas do *cluster* utilizando o padrão para comunicação MPI.

Vale destacar que o CronoSim integraliza um conjunto de trabalhos específicos que propõem melhorias no desempenho para Simulação de Eventos Discretos por meio dos conceitos de Simulação Distribuída. O componente **CronoSim Framework** especificado na arquitetura da ferramenta é resultado de diversos estudos que vêm sendo aprimorados a fim de otimizar tais melhorias. Neste contexto, este trabalho de mestrado também fornece novos mecanismos ao *framework* em questão. A seção 4.5 apresenta as contribuições propostas ao **CronoSim Framework**.

4.3 Aplicação Principal

Um tratamento visual adequado de um *software* proporciona melhor compreensão às suas funcionalidades por parte do usuário. Na arquitetura apresentada, o componente **CronoSim Application**, correspondente à camada de aplicação, tem como uma de suas atribuições apresentar os dados ao usuário por meio de uma Interface Gráfica.

As interfaces com o usuário são implementadas com base em componentes gráficos distintos, com características próprias, tais como, menus, botões, campos de texto, etc. A partir de outros componentes é possível organizá-los de forma estruturada e distribuí-los em contêineres, como painéis e janelas. Neste contexto,

a especificação desta camada da ferramenta considerou diversos recursos gráficos disponíveis nas bibliotecas nativas do Java: AWT (*Abstract Window Toolkit*) e *Swing*.

As subseções seguintes descrevem o projeto da camada de aplicação, mostrando as principais características da interface com o usuário. Serão descritas, portanto, as classes pertinentes ao editor gráfico para o processo de modelagem e também as classes responsáveis pela interação com o *cluster* durante a simulação.

4.3.1 Interface Com o Usuário

O módulo de interface do CronoSim deve cumprir os seguintes objetivos:

- Apresentar uma interface intuitiva;
- Permitir a especificação gráfica do modelo;
- Oferecer recursos para configuração das propriedades da simulação;
- Possibilitar a definição das propriedades de conexão com uma rede de computadores.

Para alcançar o primeiro objetivo, a interface com o usuário do CronoSim explorou os recursos gráficos preexistentes nas APIs (*Application Programming Interfaces*) nativas do Java, para criação de janelas, menus, barra de ferramentas, barra de rolagem, entre outros componentes básicos. Além das APIs nativas, foram utilizadas o JGraphx e o iText na construção da interface gráfica com outros recursos específicos.

4.3.2 Ambiente Para Modelagem

CronoSim oferece um ambiente para modelagem de sistemas, permitindo ao usuário descrever o modelo utilizando Redes de Filas como forma de representação. A figura 17 apresenta a interface gráfica para o ambiente de modelagem.

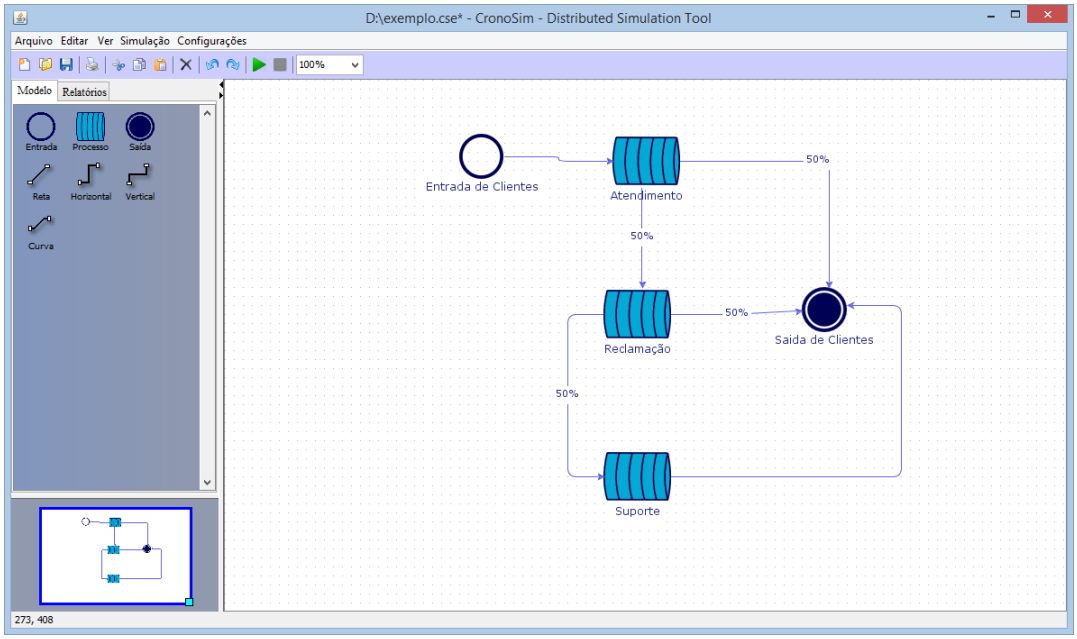

Figura 17: Ambiente gráfico de modelagem utilizando Redes de Filas

Esta área de interação consiste em uma janela com um *layout* composto pelas seguintes delimitações:

1. **Área de trabalho**: baseia-se em um painel de desenho, no qual modelos poderão ser criados e manipulados pelo usuário graficamente. Este espaço contém um tamanho adequado para este tipo de aplicação, abrangendo a parte central da janela

2. **Área de *template***: está localizada no lado esquerdo da janela e constitui as opções de componentes a serem inseridos no modelo, bem como o estilo das linhas que representam as ligações entre eles. A partir dos ícones representados nesta área é possível utilizar o recurso *drag-and-drop* para adição de novos elementos ao modelo, ou seja, para inserir um componente basta selecionar e arrastar para a área de trabalho.

3. **Área de relatórios da simulação**: permite visualizar e exportar os resultados da simulação em arquivo no formato PDF. Para acesso aos relatórios,

deve-se alternar a aba que está localizada, também, na lateral esquerda da janela.

4. ***Outline*** do modelo: consiste em um mecanismo para controle da visualização da área de trabalho. Ele facilita o gerenciamento sobre o *zoom* aplicado ao desenho do modelo, ou seja, permite ampliar ou reduzir o espaço de visualização da área de trabalho. Este item localiza-se na parte inferior esquerda do ambiente gráfico.

5. **Barra de ferramentas**: contém ícones utilizados para ativar diversas funcionalidades, que também podem ser acessadas pela barra de menus. Está localizado abaixo da barra de menus.

6. **Barra de menus**: é acoplada na parte superior da janela principal da interface gráfica e provê funcionalidades diversas à aplicação, tais como, manipulação de arquivos inerentes aos modelos desenvolvidos (abrir, criar, salvar), edição das propriedades do ambiente gráfico (exibir grades, réguas, etc), manipulação dos componentes do modelo (copiar, colar, recortar, desfazer, refazer, etc), gerenciamento da simulação (iniciar, pausar, parar) entre outras.

O modelo representado na figura 17 ilustra um exemplo da especificação de um sistema baseado em Redes de Filas. Chwif e Medina (2014) explicam algumas características desta técnica de modelagem. Os componentes básicos desta representação são: servidores, filas e clientes (eventos). Os dados modelados do sistema correspondem aos servidores, os quais prestam serviços aos clientes. A fila é um campo de espera para clientes que não são atendidos prontamente por um serviço requisitado devido aos servidores estarem ocupados. O conjunto de um ou mais servidores e uma ou mais filas é denominado centro de serviço. Nesse sentido, uma rede de filas é composta por um conjunto de centros de serviço.

Neste trabalho, os centros de serviço serão considerados processos lógicos e atuarão de forma a atender os eventos que entram no sistema. Dessa forma, utilizando o ambiente de modelagem, é possível adicionar **Processos**, **Entradas**

(geradores de eventos) e **Saída** (indica a saída do sistema). A ferramenta permite modelar processos em séries, quando existe probabilidade dos eventos, ao serem atendidos, entrarem em outro processo. Também é possível conectar processos ao componente **Saída** indicando a probabilidade dos eventos saírem do sistema.

4.3.2.1 Componentes do Modelo de Simulação

A figura 18 exibe os 3 componentes básicos para realizar a modelagem. Cada componente do modelo possui suas propriedades e o usuário pode configurá-las de acordo com o sistema que se deseja modelar.

Figura 18: Componentes para modelagem do sistema

A tabela 3 descreve as propriedades para cada componente. Elas podem ser alteradas pelo usuário ao decorrer da especificação do modelo. Para manipular estes dados, são utilizadas janelas de configurações para cada um destes elementos. A figura 19 ilustra as interfaces utilizadas durante a modelagem para alteração destas propriedades, que podem ser acessadas pelo usuário através do clique duplo sobre o componente que deseja editar.

O tipo de entidade a ser definido para os componentes do tipo **Entrada** representa os itens processados ao longo do sistema, como produtos, clientes, documentos, etc. As propriedades **tempo entre chegadas** e **tempo de atendimento** referem-se aos respectivos dados estatísticos de intervalos de chegadas e duração do serviço. Estas informações relacionadas ao tempo são definidas de acordo com a distribuição de probabilidade selecionada, ou seja, cada tipo de distribuição possui parâmetros específicos. Por exemplo, para gerar um número utilizando a distribuição exponencial é necessário somente a média, ao passo que a distribuição triangular demanda três valores: mínimo, moda e máximo.

Tabela 3: Propriedades dos componentes do modelo

Nome do Componente	**Propriedades**
Entrada	Nome Tipo de entidade Distribuição de probabilidade do tempo entre chegadas Tempo entre chegadas Unidade de Tempo
Processo	Nome Lista de recursos Distribuição de probabilidade do tempo de atendimento Tempo de Atendimento Unidade de Tempo
Saída	Nome

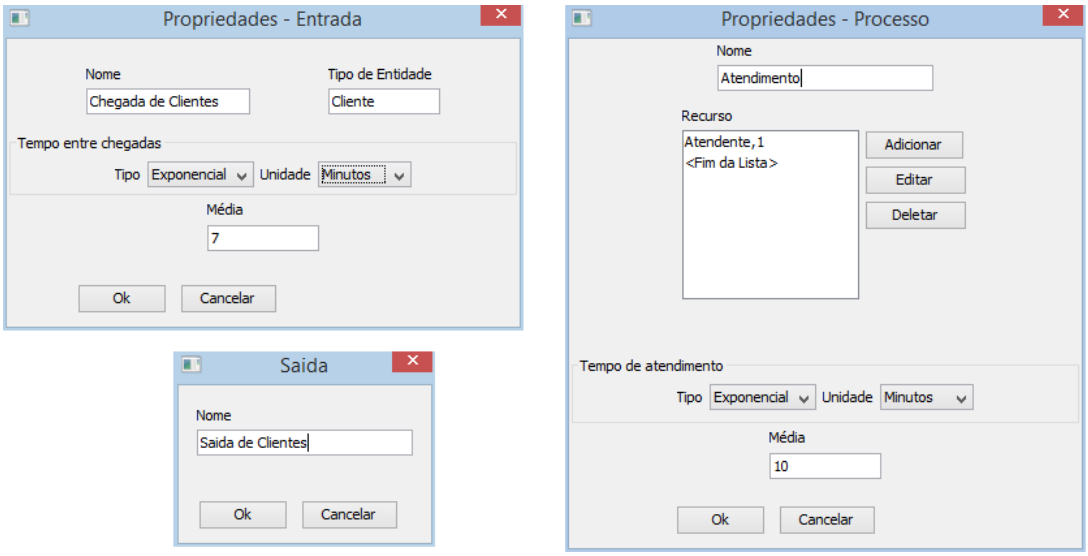

Figura 19: Configuração das propriedades dos componentes do modelo

Naturalmente, uma simulação de eventos discretos baseia-se em situações reais que ocorrem em um determinado tempo. Sendo assim, durante a modelagem, o usuário deve informar as estimativas de tempo para os componentes do modelo, conforme explicado no parágrafo anterior. Um detalhe importante ao especificar este tipo de informação é saber qual unidade de tempo se trata. Por essa razão, existe um campo pelo qual o usuário poderá determinar a **unidade de tempo** referente à cada componente do modelo.

Para todos os componentes do modelo, a propriedade **nome** é utilizada para rotular o componente no modelo, tornando o modelo organizado de forma que o usuário visualize e identifique os componentes facilmente, além de possibilitar a análise dos relatórios da simulação após a geração dos resultados.

4.3.2.2 Representação de Fluxos

A representação das transições das entidades entre os processos do sistema consiste em setas direcionadas que interligam a origem e destino. A figura 20 ilustra um modelo simples, que possui um único processo: **Atendimento**. Neste sistema, a chegada de clientes é definida por um bloco de entrada e, a partir deste bloco, a entidade é gerada e enviada ao processo que o atenderá. Por fim, após o processamento do serviço, a entidade sairá do sistema, como pode ser observado no exemplo.

Figura 20: Representação de fluxo nos modelos

Em sistemas de eventos discretos, uma entidade pode conter diferentes alternativas de transição entre os processos envolvidos. Neste caso, considera-se uma tomada de decisão durante o processamento da entidade. Um exemplo disto, pode ser observado na figura 21, que representa um processo no setor industrial, no qual uma máquina de inspeção de peças realiza a análise considerando duas opções possíveis: peças boas ou peças refugadas.

O projetista da simulação deve informar as chances de ocorrências para cada alternativa. Existe, também, a possibilidade de entidades voltarem ao final da fila do mesmo processo, pelo qual completou seu atendimento. Neste caso, a transição pode ser representada no editor gráfico utilizando um *loop*, como ilustrado na figura 22.

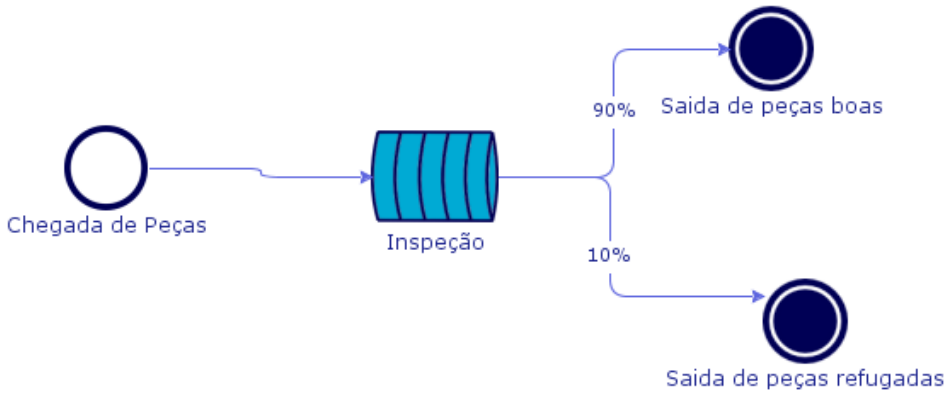

Figura 21: Representação de fluxo alternativos

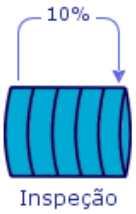

Figura 22: Representação de transições de entidades em *loop*

Dessa forma, pode-se especificar diversos fluxos de saída a partir de um mesmo processo do modelo e definir as respectivas probabilidades. A interface utilizada para alterar estes valores é ilustrada na figura 23. O sistema valida estas probabilidades de modo que a soma de todos os valores deve ser igual a 100, uma vez que representa as chances de ocorrência de cada situação. Nos casos em que existe apenas um fluxo simples, como no exemplo da figura 20, o valor sempre será 100% e, portanto, fica implícito no editor gráfico.

4.3.3 Parâmetros da Simulação

Além do ambiente para modelagem, a interface gráfica fornece outros recursos para a construção de uma simulação. Após definir completamente o modelo, a

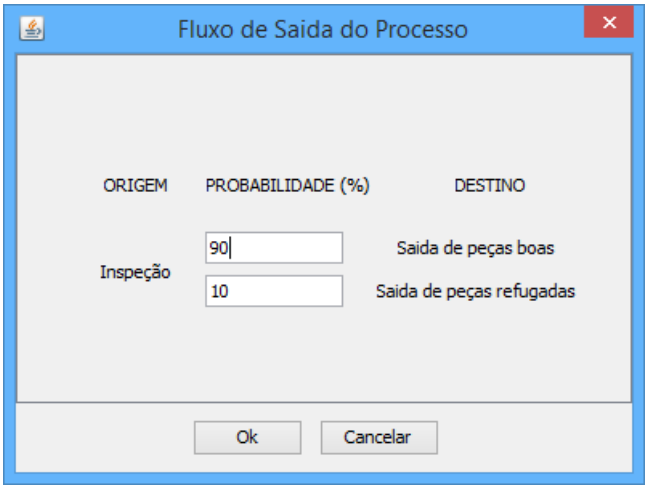

Figura 23: Interface para definição das probabilidades de cada fluxo

próxima etapa é especificar algumas propriedades inerentes à execução da simulação. Para isto, a ferramenta apresenta outra perspectiva que permite a definição destas informações, como ilustra a figura 24.

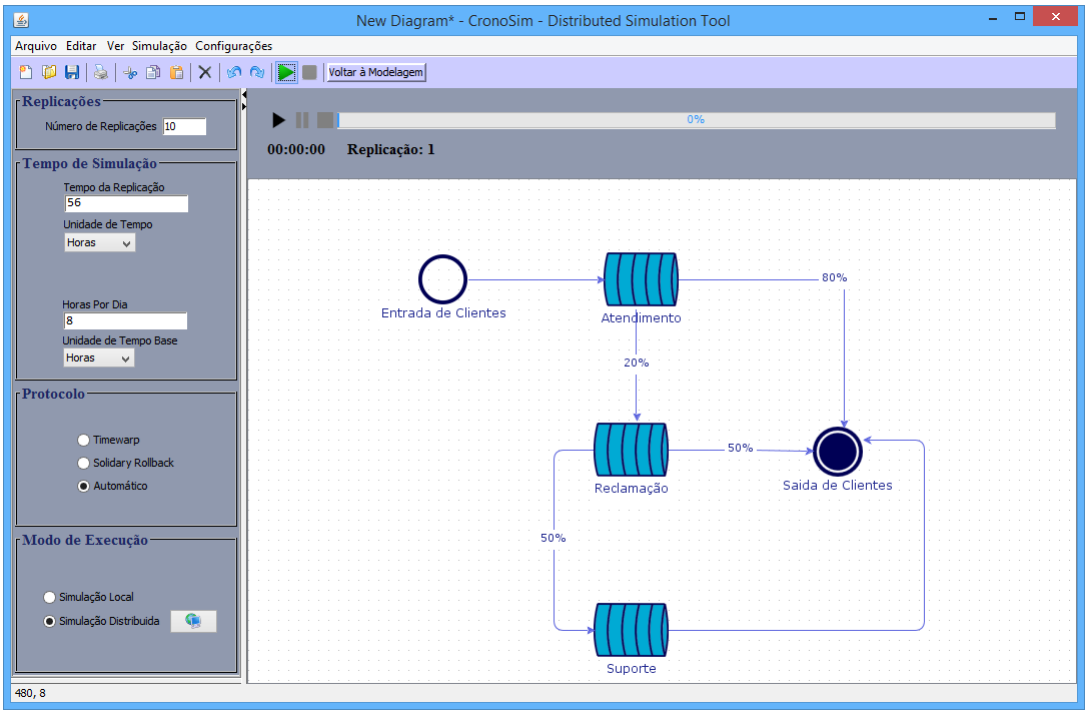

Figura 24: Tela para definição das propriedades de execução da simulação

As ferramentas de simulação, em geral, permitem que o usuário especifique como será a rodada, determinando a duração da simulação e o número de replicações. Segundo Chwif e Medina (2014), a rodada refere-se ao que ocorre quando seleciona-se ou inicia-se o comando que executa a simulação no computador. Uma rodada pode envolver várias replicações. Dessa forma, cada rodada do modelo deve ser entendida como um novo "experimento".

Portanto, antes de iniciar a execução de uma rodada da simulação, estes dados devem estar configurados corretamente de acordo com as características do sistema a ser simulado. Os itens a seguir descrevem estes dados:

- **Número de replicações**: uma replicação é uma repetição da simulação do modelo, com a mesma configuração, a mesma duração e com os mesmos parâmetros de entrada, mas com uma semente de geração dos números aleatórios diferente (CHWIF; MEDINA, 2014).

- **Tempo de simulação**: representa o período de operação correspondente ao sistema real que se deseja simular. Por exemplo, um supermercado que fica aberto das 9 às 22 horas deve ser simulado por um período de 13 horas (CHWIF; MEDINA, 2014). Sendo assim, pode-se determinar, também, o tempo diário de funcionamento do sistema. Além disso, é possível especificar a unidade de tempo base dos valores que serão mostrados no relatório.

- **Protocolo**: refere-se ao protocolo de sincronização dos processos na simulação distribuída. Entre as opções disponíveis estão: *Time Warp* e *Rollback* Solidário.

- **Modo de execução**: apesar do foco deste trabalho ser a execução distribuída da simulação, a ferramenta permite tratar localmente o processamento. Desta maneira, é possível realizar simulações usando apenas o próprio computador do usuário. Mesmo assim, a ferramenta realiza o particionamento da simulação quando há multiprocessadores, tornando o desempenho melhor. Para o modo "simulação distribuída", o usuário deverá informar o domínio do

servidor do *cluster* que servirá como plataforma de execução. É importante destacar que o *cluster* em questão deverá estar previamente configurado para que seja possível realizar a execução. A seção 4.4 apresentará a aplicação responsável por tratar a simulação no lado do servidor.

4.3.4 Relatórios da Simulação

Uma das funcionalidades mais relevantes das ferramentas de simulação é a apresentação dos resultados. CronoSim fornece recursos para gerar relatórios dos resultados após a execução dos modelos. É possível exportar estas informações para um arquivo PDF (*Portable Document Format*) para melhor análise dos resultados. A figura 25 exibe a interface para visualização dos dados resultantes de uma simulação.

CronoSim oferece 2 tipos de visualizações específicas dos resultados da simulação, como listados a seguir:

1. **Relatório geral:** apresenta os resultados considerando o experimento completo, ou seja, utiliza a média de todas as replicações realizadas. Os resultados são apresentados para todos os processos e saídas do modelo. Um exemplo deste relatório pode ser observado na figura 25, no qual mostra os resultados para um modelo com apenas um processo (Atendimento) e uma saída (Saída de Clientes). Este tipo de relatório é o mais completo, sendo que os demais tipos de visualização dos relatórios representam partes específicas ou mais detalhadas do relatório geral.

2. **Relatório por replicação:** similar ao relatório geral, no entanto, mostra os valores de cada replicação separadamente.

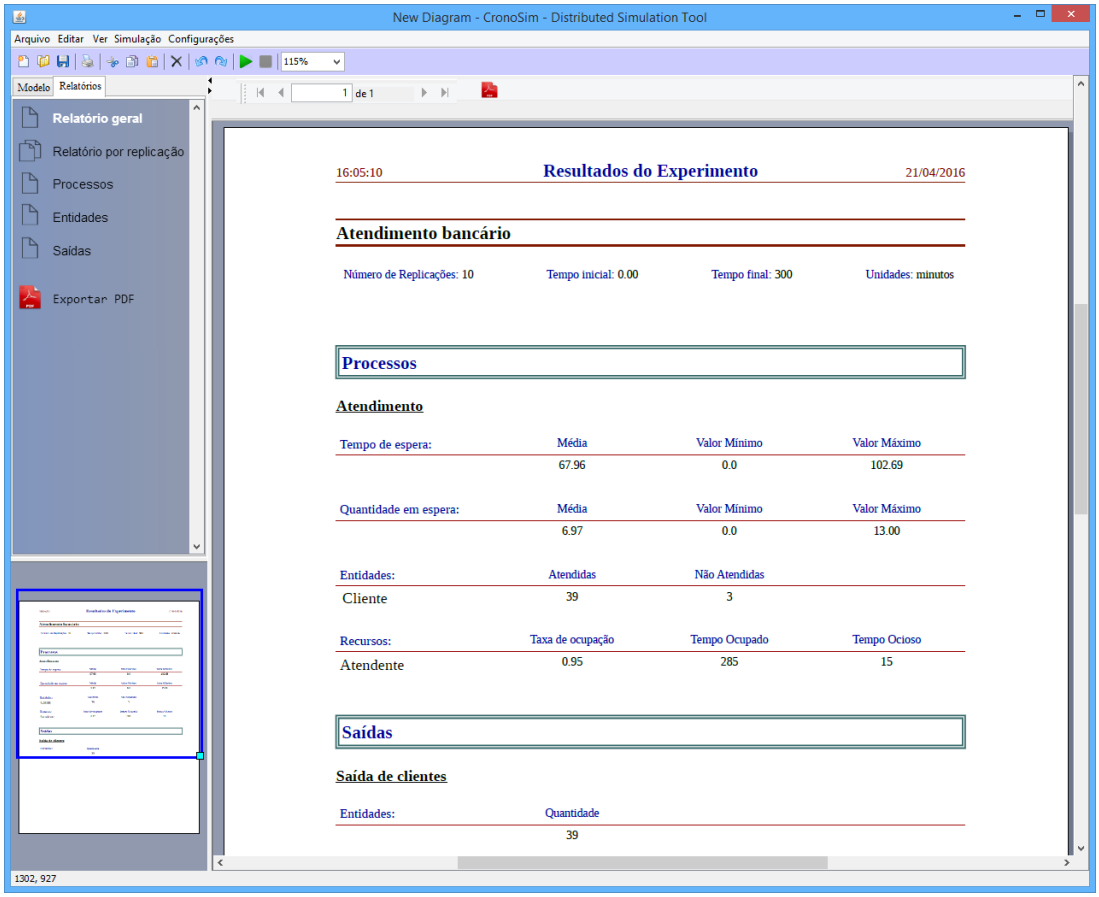

Figura 25: Relatórios do CronoSim

4.3.5 Especificação da Aplicação Principal em UML

A aplicação principal é composta por pacotes de classes que agrupam funcionalidades específicas. Esta separação é uma característica considerável de projetos orientados a objetos e propicia melhor organização, facilitando o desenvolvimento, uma vez que a manutenção de código em determinado pacote não interfere nos demais. Os itens listados a seguir apresentam os pacotes que compõem o projeto da aplicação principal.

1. *unifei.edu.cronosim.editor*: contém as classes responsáveis pela construção de todo ambiente gráfico da ferramenta. O arcabouço da interface gráfica

é construído a partir de componentes nativos das bibliotecas *Swing* e AWT (nativas da linguagem Java), JGraphx e iText.

2. *unifei.edu.cronosim.views*: assim como o pacote "editor", este pacote também compreende recursos gráficos. Ele consiste em classes para exibição das caixas de diálogo para definição das propriedades de cada componente do modelo.

3. *unifei.edu.cronosim.actions*: abrange as classes responsáveis por tratar as ações que são ativadas pelos comandos do usuário. Neste caso, a maioria das funcionalidades da ferramenta são realizadas a partir da ativação de uma determinada ação. Os comandos estão associados a diversos elementos dispostos na ferramenta, os quais o usuário gerencia. Por exemplo, a barra de menus e a barra de ferramentas contêm itens que realizam uma determinada ação no sistema. Quando o usuário seleciona uma destas opções, uma ação é acionada e um procedimento específico é realizado.

4. *unifei.edu.cronosim.simulation*: contém as classes inerentes à execução da simulação, localmente ou através de um servidor externo. Envolve funcionalidades correspondentes à comunicação com o servidor, bem como o tratamento de mensagens entre o *cluster* e o computador do usuário.

5. *unifei.edu.cronosim.images*: este pacote não possui classes do projeto, mas contém as imagens e ícones que são aplicados na interface gráfica. As classes envolvidas na construção gráfica da ferramenta faz uso deste pacote.

6. *unifei.edu.cronosim.resources*: possui arquivos que definem algumas propriedades do sistema, como o estilo dos recursos gráficos (tipos e tamanhos das fontes dos textos, cores, e diversas características), gerenciamento do idioma da ferramenta e controle sobre os dados de conexão com um servidor. São utilizados arquivos com a extensão XML (*eXtensible Markup Language*) e arquivos de propriedades (*.properties*), que permitem a configurações para uma determinada aplicação.

Os diagramas de classes dos pacotes correspondentes à aplicação principal serão apresentados nas próximas subseções.

4.3.5.1 Pacote "Editor"

A figura 26 mostra a relação entre as principais classes do pacote **editor**, destacando a utilização de recursos das bibliotecas gráficas. Neste contexto, as classes nativas da linguagem Java, como **JFrame**, **JPanel**, **JToolBar**, **JMenuBar**, **JPopupMenu**, foram empregadas. Além disto, a biblioteca JGraphx é explorada, sendo utilizadas instâncias de classes, como **mxGraphOutline**, **mxGraphComponent**, **mxUndoManager** e **mxKeyBoardHandler**. Diversas classes foram desenvolvidas neste projeto, herdando este conjunto de recursos.

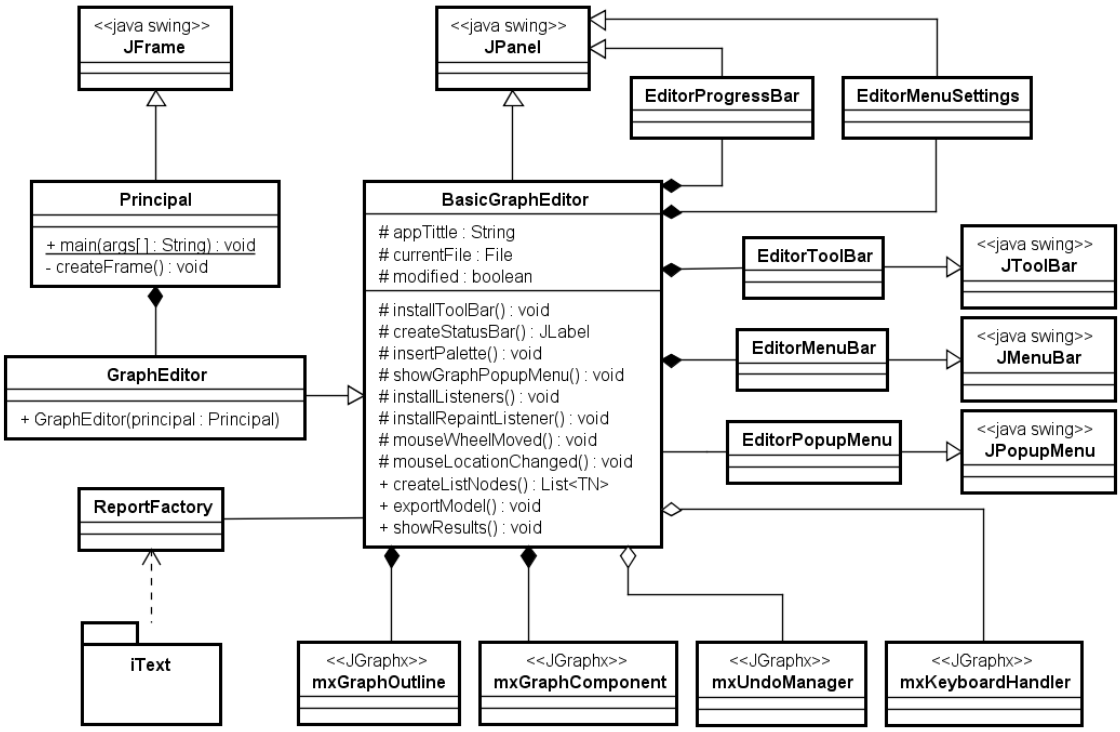

Figura 26: Diagrama de classes do pacote **editor**

Neste pacote, a classe **Principal** contém a função *main*, responsável por dar

início a execução da ferramenta. Ela constrói a janela principal do CronoSim, que envolve todos os demais componentes gráficos. Para tanto, esta classe estende todas as características e comportamentos de **JFrame**. O método **createFrame**, inicializa as propriedades da janela e adiciona o painel que contém todos os demais recursos da ferramenta. Este painel é representado pela classe **GraphEditor**, que por sua vez, é uma especialização da classe **BasicGraphEditor**.

Além da construção dos componentes gráficos correspondentes à estrutura da janela, são utilizados recursos da biblioteca JGraphx para construção dinâmica dos modelos de simulação. As funcionalidades oferecidas por esta API, possibilitaram a elaboração de um mecanismo para desenho de modelos de simulação, conforme explicado na seção 4.3.2. A construção dos modelos baseia-se em uma estrutura fornecida pela classe **mxGraphComponent**, a qual representa o componente onde os elementos podem ser adicionados e manipulados de forma simples. Por meio disto, é possível controlar os elementos no ambiente de modelagem utilizando recursos intuitivos e realizar ações como: selecionar elementos, arrastar (*drag-and-drop*), copiar, colar, recortar, mover, etc.

A vantagem de utilizar as implementações da biblioteca JGraphx é a reutilização de diversas funções complexas, que demandariam muito tempo para desenvolvê-las. Sobretudo, outra funcionalidade apresentada por esta classe é a geração de um arquivo XML que contém os dados sobre o desenho construído, neste caso, correspondente ao modelo de simulação. A listagem 4.1 mostra um trecho de um arquivo XML gerado a partir de um modelo. Apesar das funções responsáveis pela geração deste arquivo serem implementadas pelo JGraphx, é possível adaptar a saída dos dados gerados no arquivo.

Listagem 4.1: Exemplo da estrutura de um arquivo XML gerado a partir do modelo

```
1 <mxCell id="13" parent="1" style="image;image=/edu/unifei/cronosim/images/input.png"
      vertex="1">
2 <TInput as="value" idSim="E1" name="Entrada de Clientes">
3 <TimeDatas as="timeDatas" delayDistribution="Constant" delayTimeValue="10"
      timeUnit="Minutes" />
```

```
4 </TInput>
5 <mxGeometry as="geometry" height="50.0" width="50.0" x="180.0" y="100.0" />
6 </mxCell>
```

Sendo assim, neste exemplo, a *tag* **TInput** foi integrada ao modelo XML gerado para representar um componente do tipo **Entrada**. A *tag* **mxCell** representa um elemento gráfico do desenho (neste caso, para um componente do tipo **Entrada**). Ela possui as *subtags* **TInput** e **mxGeometry** que especificam outras informações a respeito deste componente, como os dados referentes ao modelo (tempo entre chegadas, distribuição de probabilidade, unidade de tempo, etc) e os dados relacionados ao tamanho e posição do desenho.

O arquivo XML gerado corresponde ao sistema modelado pelo usuário. Portanto, ao desenvolver um modelo, o usuário pode salvá-lo em um arquivo para que seja possível abri-lo novamente. Apesar da estrutura do arquivo resultante ser estruturado em XML, a ferramenta o gera com uma nova extensão, "CSM" (abreviação de *CronoSim Model*).

Outras classes da biblioteca JGraphx compõem o projeto, como a **mxGraphOutline**, responsável por gerar o componente "*Outline*", apresentado na seção 4.3.2, além das classes: **mxUndoManager** e **mxKeyBoardHandler**. A primeira está relacionada aos mecanismos de controle sobre alterações no modelo ("desfazer" e "refazer"), ao passo que a segunda trata ações através das teclas de atalho para melhor manipulação do modelo.

A classe **BasicGraphEditor** envolve a maior parte das funcionalidades de construção do ambiente gráfico. Ela é uma extensão de **JPanel** e, portanto, é composta por diversos outros componentes gráficos que, juntos, formam o ambiente de simulação. Por exemplo, as classes **EditorProgressBar** e **EditorMenuBar**, representam outros painéis, os quais constroem arcabouços para a barra de progressos utilizada durante a simulação e o formulário com os parâmetros de configuração da simulação, respectivamente. Da mesma forma, existem outras classes responsáveis por construir partes dos componentes gráficos. Dentre elas, tem-se

a classe **EditorToolBar**, para barra de ferramentas; **EditorMenuBar**, para a barra de menus e; a classe **EditorPopupMenu**, para exibição de menus *popup*. Estas três últimas são especializações de classes da biblioteca *Swing*.

Os atributos **appTittle**, **currentFile**, **modified** estão relacionados ao modelo que o usuário edita. Eles representam o nome, o arquivo que será armazenado e o *status* (que indica se as últimas modificações no modelo foram salvas) para o modelo em desenvolvimento.

As operações **installToolBar**, **createStatusBar**, **insertPallete**, **showGraphPopupMenu** e **installRepaintListener** servem para separar a especificação dos componentes gráficos, como barra de ferramentas, barra de *status*, paleta de *templates* (para inserção de elementos do modelo), exibição de um menu *popup* disponível na tela principal e, também, a inicialização de métodos ouvintes dos comandos relacionados ao efeito *drag-and-drop* destes *templates* para a área de trabalho do ambiente de modelagem.

As operações **mouseWheelMoved** e **mouseLocationChanged** são implementações correspondentes às interfaces nativas do Java **MouseWheelListener** e **MouseMotionListener**. Elas são responsáveis por tratar eventos ativados pelo *mouse*, realizando ações específicas de acordo com o comando efetuado. O método **installListeners** adiciona outros ouvintes de eventos de *mouse*, que tratam os comandos realizados sobre o desenho do modelo na área de trabalho da ferramenta.

Os métodos **createListNodes**, **exportModel** e **showResults** envolvem o tratamento dos dados após a modelagem. Quando um modelo é finalizado e o procedimento de execução se inicia, a operação **createListNodes** é invocada para que os componentes do modelo sejam transformados em uma lista de objetos que serão serializados. Estes objetos são instâncias da classe **TemplateNodes** disponível no novo pacote implementado no **CronoSim Framework**, que será explicado mais adiante (no diagrama utilizou-se a abreviação “TN” por questões de espaço).

O usuário deve, também, fornecer os parâmetros da simulação, como o número

de replicações, tempo da simulação e modo de execução, conforme explicado na seção 4.3.3. De posse destas informações e, também, da lista de objetos correspondente ao modelo, a operação **exportModel** gera um arcabouço do projeto de simulação para ser enviado ao módulo de execução. Neste caso, esta exportação resulta em um objeto serializado da classe **ModelDatas**, disponível no **CronoSim Framework**.

Por fim, o método **showResults** é responsável por tratar os resultados recebidos após o término da simulação e invocar os métodos de geração de relatórios disponíveis na classe **ReportFactory**. Esta classe, por sua vez, utiliza recursos da biblioteca iText na construção de documentos textuais estruturados.

4.3.5.2 Pacote "Views"

A figura 27 ilustra o diagrama de classes do pacote *views*, que contém as classes responsáveis por permitir a especificação dos dados de cada componente do modelo. As 4 classes deste pacote são subclasses de **JDialog**, para exibição de uma janela de diálogo. Estas janelas contém uma espécie de formulário a respeito dos elementos do modelo. Todas, também, têm em comum 3 métodos, uma vez que elas implementam a interface **ViewComponent**. No entanto, estes métodos realizam as operações de forma diferente em cada uma, ou seja, operam de acordo com os dados de cada elemento do modelo.

Conforme foi explicado sobre o ambiente de modelagem, o usuário pode alterar os dados de um componente efetuando o duplo clique sobre ele. Assim, a janela com o formulário será exibida, sendo possível a alteração. O método **initComponents** é responsável por construir o formulário para cada tipo de componente, pelo qual o usuário fornecerá os dados. A operação **loadFields** preenche os campos do formulário com os dados atuais do elemento selecionado, enquanto a operação **saveFields** salva os dados após o usuário concluir o preenchimento.

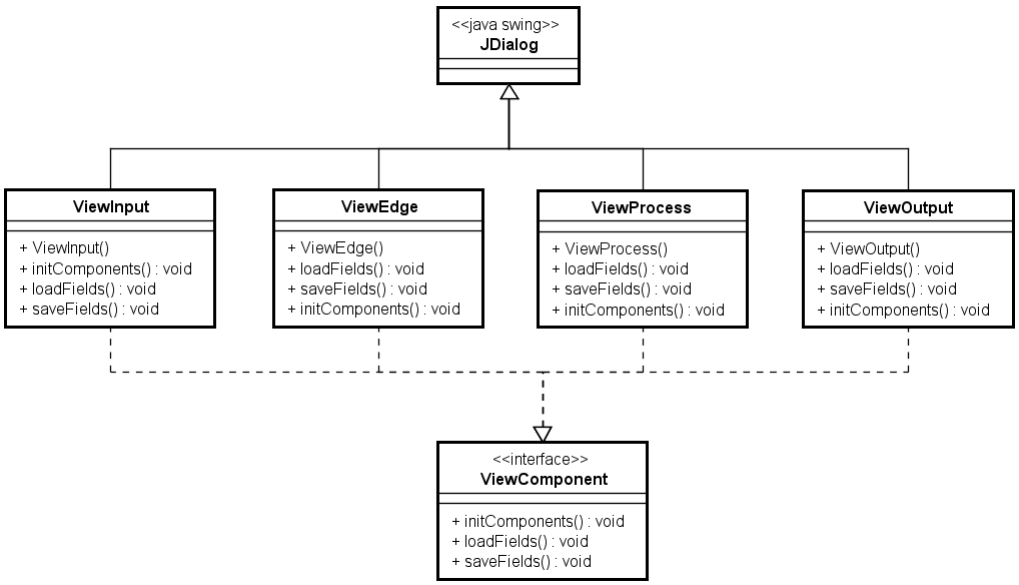

Figura 27: Diagrama de classes do pacote **views**

4.3.5.3 Pacote "Actions"

As classes implementadas neste pacote são responsáveis por tratar as ações ativadas por diversos comandos do usuário. Os métodos destas classes estão associados às opções da barra de ferramentas, barra de menus e outros botões dispostos na ferramenta. A tabela 4 mostra as classes implementadas neste pacote com suas respectivas atribuições.

4.3.5.4 Pacote "Simulation"

Este pacote abrange as classes responsáveis por tratar da execução da simulação. As classes que compõem este pacote são, basicamente, **ClusterCommunication** e **LocalSimulation**, como ilustra a figura 28. O relacionamento delas é realizado entre outras classes relacionadas à aplicação do servidor. Ou seja, existe uma interação deste pacote com classes implementadas no módulo **CronoSim Server** que será apresentado adiante.

Tabela 4: Classes do pacote *Actions*

Classes	**Atribuições**
NewAction	Criar novo modelo. Neste caso, é verificado se o modelo atual está salvo para limpar a área de trabalho.
SaveAction	Salvar o modelo atual no formato CSM (*CronoSim Model*).
OpenAction	Abrir um modelo no formato CSM.
StartAction	Iniciar a simulação.
PauseAction	Pausar a simulação após seu início. Ao efetuar a pausa é possível ver relatórios parciais de uma simulação.
ContinueAction	Continuar uma simulação que foi pausada.
StopAction	Interromper uma simulação. Neste caso, os resultados irão considerar a simulação até o momento em que foi interrompido.
SwitchPerspectiveAction	Alterar a perspectiva de visualização da ferramenta. A ferramenta apresenta 3 perspectivas básicas, contendo os ambiente de: modelagem, execução e relatórios.
GridColorAction	Alterar cores da interface gráfica.
GridSlyleAction	Configurar exibição de grades na área de trabalho.
PageSetupAction	Configurar as dimensões da "página" correspondente à área de trabalho.
ZoomPolicyAction	Configurar a ampliação da área de trabalho.
SetLabelPositionAction	Mostrar as coordenadas do *mouse* em relação ao posicionamento na área de trabalho.
FontStyleAction	Modificar fontes na ferramenta.
HistoryAction	Gerenciar o histórico das ações realizadas a fim de permitir utilizar os comandos "refazer" e "desfazer".
WarningAction	Informar quando existe uma inconsistência no modelo. Ou seja, durante a modelagem, o sistema realiza verificações de algumas regras definidas para criação dos modelos. Por exemplo, não é possível efetuar conexões entre dois componentes do tipo **Entrada**.
ExitAction	Realizar o procedimento para fechar a ferramenta. A verificação se o modelo atual está salvo é realizada.

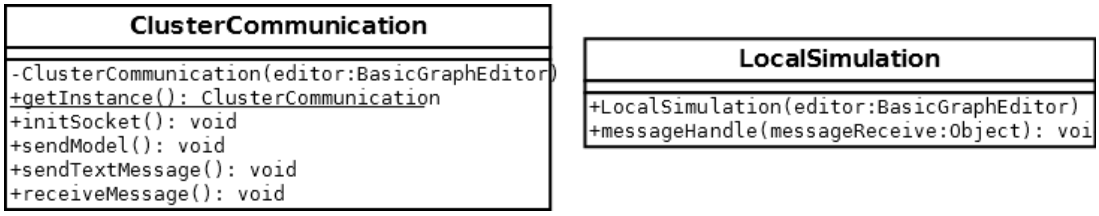

Figura 28: Classes do pacote **simulation**

4.4 Aplicação no Servidor

O projeto da ferramenta considerou a implementação de um módulo para gerenciamento da simulação no sistema distribuído. Este módulo corresponde ao componente denominado **CronoSim Server**, conforme introduzido no início deste capítulo. Uma das suas principais atribuições é o tratamento da conexão entre a aplicação principal e o *cluster* (em que está hospedado).

A aplicação no lado do servidor, bem como o *cluster* utilizado, possuem as seguintes características:

- Um computador (*front-end*) é responsável por gerenciar as conexões com aplicações de usuário e monitorar as simulações.
- O *front-end* deve estar executando a aplicação **CronoSim Server**, que funcionará como um serviço. Além disso, existe um diretório (*cronosim_home*), no *front-end*, que é compartilhado por todas as máquinas do *cluster*. Este compartilhamento é realizado por um sistema de arquivos distribuídos, utilizando o NFS (*Network File System*) (COULOURIS et al., 2013). Portanto, todas as máquinas do *cluster* têm acesso a este diretório, específico para controle das simulações.
- A cada nova conexão, um novo *server socket* é iniciado por uma *thread*.
- Após receber a mensagem para início de uma nova simulação, a aplicação no servidor gera um diretório temporário (*user_home*) para salvar arquivos de

controle sobre a simulação. Existirá um diretório com identificador exclusivo para cada simulação. As informações armazenadas neste diretório, portanto, serão utilizadas por um mecanismo de monitoramento durante a execução e, também, para serializar os resultados, que serão enviados à aplicação principal. Depois de enviar os resultados, o diretório é deletado.

- Quando o servidor aceita uma conexão pelo *socket*, o procedimento de execução da simulação é iniciado. Então, o programa invoca métodos disponíveis no **CronoSim Framework** para inicialização da simulação nas máquinas do *cluster*. Para tanto, utiliza-se o MPJ-Express por meio de comandos de sistema.

- Após este procedimento, uma *thread* é disparada para controlar o andamento da simulação. A *thread* em questão é constituída por um *loop* que faz o monitoramento sobre a evolução da simulação em determinados períodos de tempo (utilizando as informações armazenadas no diretório *user_home*). Para isto, utiliza-se o método **Thread.sleep**, disponível na linguagem Java, que faz com que o seguimento atual da execução desta *thread* fique suspenso por um tempo determinado. Isto evita a sobrecarga de processamento neste *loop* do programa, suspendendo por alguns milissegundos.

O diretório *cronosim_home* é um elemento relevante à aplicação, sendo utilizado para armazenamento de arquivos de controle durante a simulação. Todavia, é necessário configurá-lo adequadamente para garantir o funcionamento dos mecanismos de monitoramento sobre ele. Para tanto, algumas configurações devem ser realizadas previamente no *cluster*, tais como:

1. Criação do diretório, no *front-end*, para ser utilizado pela aplicação, por exemplo:

   "/home/cluster/cronosim_home"

2. Criar uma variável de ambiente para o caminho deste diretório:

"CRONOSIM_HOME"

3. Realizar as configurações 1 e 2, nas demais máquinas do *cluster*, utilizando o mesmo nome do diretório e da variável de ambiente. Após isto, compartilhar o diretório criado no *front-end* com todas as outras máquinas através de um servidor NFS. O diretório compartilhado deve ser montado, nas máquinas *back-end*, no diretório com o mesmo caminho do original, como ilustra a figura 29.

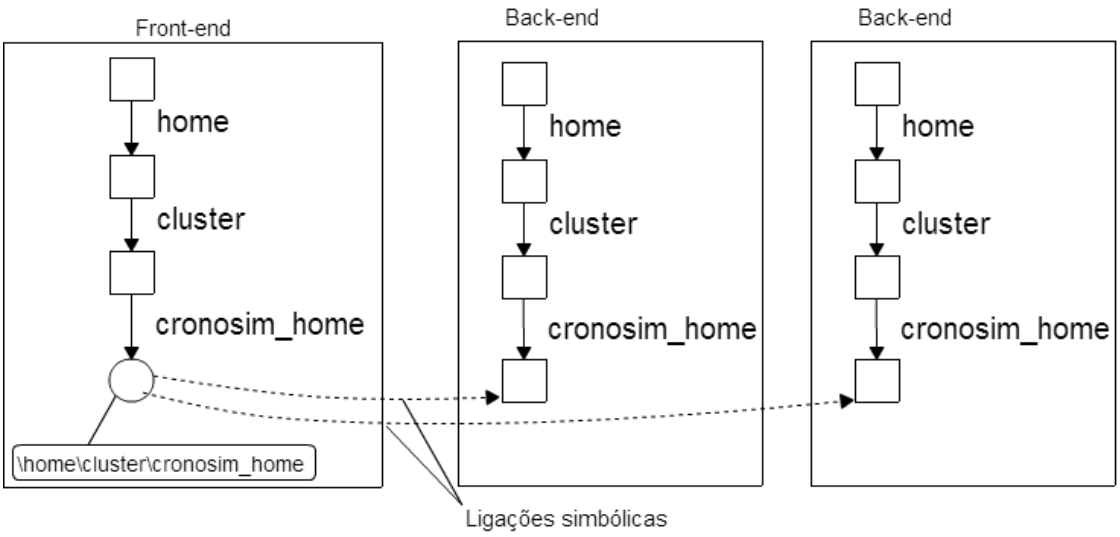

Figura 29: Compartilhamento do diretório *cronosim_home* através de um sistema de arquivos distribuídos

Com isto, será possível à aplicação criar, para cada simulação, um diretório "*user_home*" (com um identificador específico) dentro de "*cronosim_home*". Este, por sua vez, irá armazenar os seguintes arquivos durante a execução:

1. **model**: é um arquivo com os dados serializados que representam o modelo a ser simulado.

2. **machines**: contém a lista dos endereços IP (*Internet Protocol*) de todas as máquinas que serão utilizadas na simulação distribuída.

3. **arquivos de *status***: servirão para notificar alguma modificação no *status* da simulação. A partir destes arquivos é possível identificar o estado da simulação. As situações possíveis que podem ocorrer durante o procedimento de execução são: “executando”, “finalizado”, “cancelado”, “pausado”.

O componente **CronoSim Server** foi projetado para atender, também, simulações locais, ou seja, sem uso de um *cluster*. Para isto, este módulo foi acoplado à aplicação principal da ferramenta, possibilitando que a execução seja realizada no próprio computador do usuário. Neste caso, funciona como uma biblioteca, que fornece os métodos para gerenciamento da simulação. A subseção seguinte apresenta a especificação deste componente.

4.4.1 Especificação da Aplicação no Servidor em UML

O diagrama de classes do componente **CronoSim Server** é apresentado na figura 30. Pode-se observar, também, a relação com as classes da aplicação principal, que são responsáveis por tratar a comunicação com o servidor ou iniciar a execução localmente. Conforme explicado na seção 4.3, as funcionalidades envolvidas com o módulo de execução estão nas classes **ClusterCommunication** e **LocalSimulation**, ambas compõem o pacote *simulation*.

O componente **CronoSim Server** fornece recursos que permitem o gerenciamento de uma simulação utilizando o **CronoSim Framework**, tanto em um servidor externo quanto localmente. Além disto, funciona como uma biblioteca que pode ser adicionada em qualquer projeto para tratar a execução utilizando o *framework* em questão. Diante disto, à aplicação principal acopla este pacote como uma biblioteca para execução local. As próximas subseções explicam as duas maneiras de utilização do pacote *cronosimserver*.

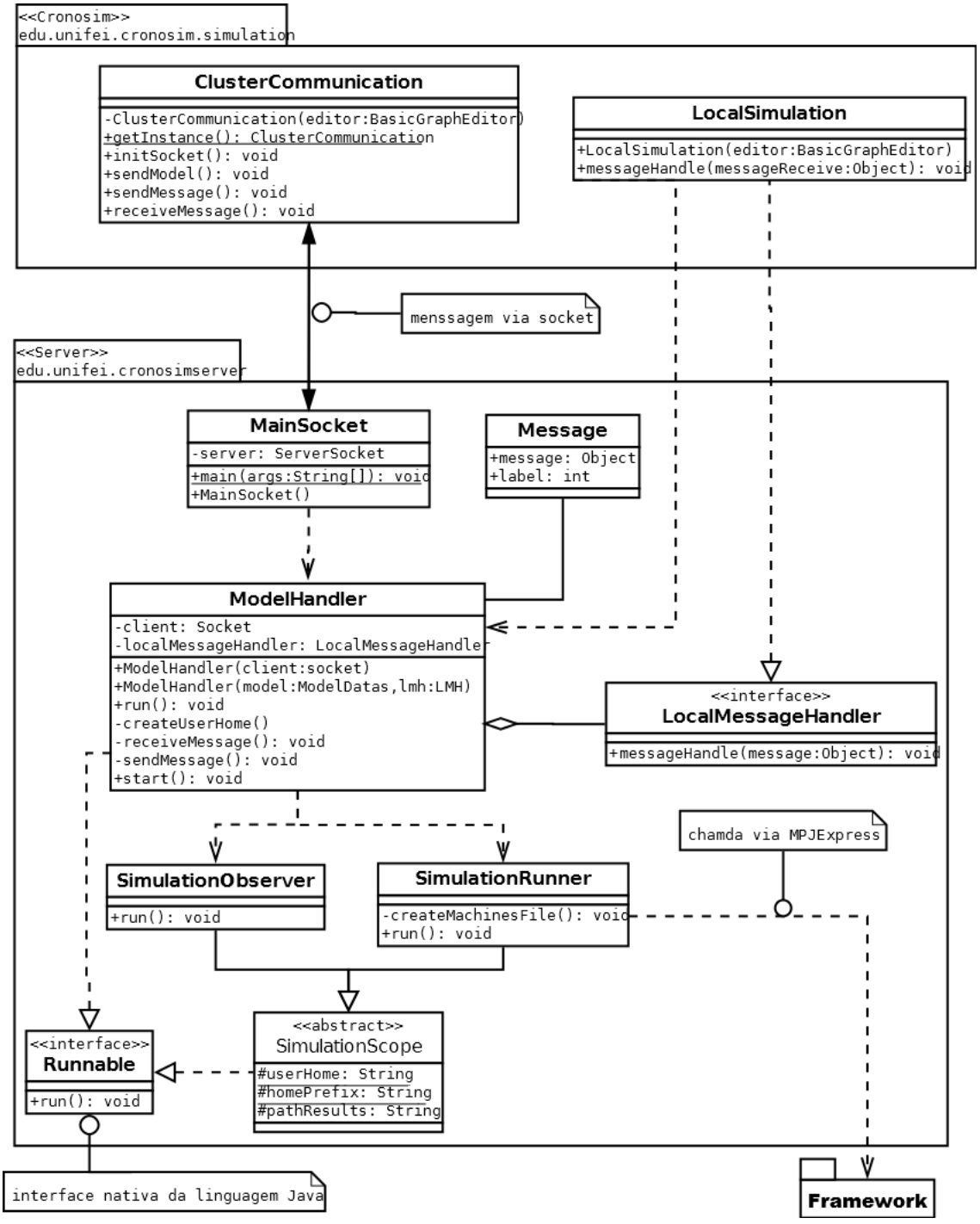

Figura 30: Diagrama de classes do pacote CronoSim Server - responsável por iniciar execução da simulação

4.4.1.1 Simulação no Servidor

A classe **ClusterCommunication** é responsável por tratar a conexão entre aplicação do usuário e o servidor. Ela é implementada utilizando o padrão de projeto *Singleton*, fazendo com que exista somente uma instância durante toda a execução. Quando o usuário efetua o comando para dar início a uma simulação distribuída, os dados do modelo e parâmetros do experimento são armazenados em uma estrutura específica para ser tratada pelo **CronoSim Server**. Para geração desta estrutura é utilizada uma classe "serializável" que é implementada no *framework*. A seção 4.5 apresentará as novas estruturas disponíveis no **CronoSim Framework**, as quais foram desenvolvidas, também, neste trabalho.

Neste contexto, a comunicação com a aplicação no servidor é iniciada por meio do método **initSocket**. Uma vez estabelecida a comunicação, o método **sendModel** envia ao servidor (*front-end* do *cluster*), uma mensagem através do *socket*. A construção da mensagem é realizada utilizando a classe **Message** do pacote *cronosimserver*. Nesta classe, existe o atributo **message**, que refere-se a mensagem, propriamente dita, sendo representada pelo tipo **Object**. Assim, é possível associar um objeto de qualquer tipo a este atributo. Uma mensagem inclui, também, um rótulo para distinguir os tipos de mensagens, representado pelo atributo **label**. A tabela 5 apresenta os rótulos utilizados nas mensagens da aplicação.

A função **sendModel** envia uma mensagem ao servidor contendo o modelo a ser simulado e, portanto, com o rótulo "*MODEL*". Os demais tipos de mensagens são enviados ao *cluster* através do método *sendMessage*. Todas as mensagens recebidas do servidor são tratadas pelo método **receiveMessage**.

No pacote *cronosimserver*, tem-se a classe **MainSocket** que inicia o *socket* servidor para disponibilizar o serviço. Ao receber uma conexão, é iniciada uma *thread*, encarregada de atendê-la, enquanto o *socket* servidor ficará aguardando outra conexão, garantindo, assim, a disponibilidade do serviço.

Tabela 5: Rótulos utilizados nas mensagens

ID	**Rótulo**	**Descrição do conteúdo da mensagem**
0	*MODEL*	Representa o modelo e parâmetros do experimento.
1	*CANCEL*	Indica o cancelamento da simulação.
2	*PAUSE*	Utilizada suspender a execução até que haja um comando para continuar ou finalizar.
3	*CONTINUE*	Quando a simulação está pausada, uma mensagem com este rótulo é utilizada para dar continuidade à execução.
4	*STOP*	Utilizada para indicar que a simulação foi finalizada.
5	*MONITOR*	Representa dados referentes ao monitoramento dos processos nas máquinas do *cluster*.
6	*ERROR*	Uma mensagem com este rótulo é disparada quando ocorre algum erro durante o processamento e a simulação não pode continuar.
7	*RESULTS*	Representa os resultados da simulação.
8	*PARTIAL*	Representa os resultados parciais da simulação. Durante a simulação, é possível obter resultados antes do tempo estipulado. Neste caso, os resultados são enviados com este rótulo.

Esta *thread* utiliza a classe **ModelHandler** como executora do serviço, visto que esta classe implementa a interface **Runnable**. Deste modo, quando a *thread* é disparada, o método **run** implementado em **ModelHandler** é chamado. As ações realizadas neste procedimento estão relacionadas com o recebimento dos dados do modelo e a preparação do diretório de controle da simulação (*user_home*). Para isto, é invocado o método **receiveMessage**, responsável por tratar todas as mensagens recebidas, sendo que a primeira delas contém os dados do modelo. Em seguida, o método **start** é chamado para dar início à simulação. A subseção 4.4.1.3 apresentará este mecanismo de inicialização.

4.4.1.2 Simulação Local

A simulação local refere-se à execução no próprio computador onde é realizada a modelagem pelo usuário. No lado da aplicação principal, a classe **LocalSimulation** é responsável por iniciar a simulação localmente. Neste caso, também é necessário a utilização dos métodos implementados no módulo **CronoSim Server**. No entanto, como foi explicado, este componente é executado no servidor como um serviço, a partir de *sockets*. Para possibilitar sua utilização em simulações locais, o projeto do pacote *cronosimserver* considerou a implementação de recursos que permitem mais flexibilidade, tais como: métodos e construtores com parâmetros específicos, além do uso de interface.

Para este modo de execução, o pacote *cronosimserver* é utilizado como biblioteca e a interface **LocalMessageHandler** permite que uma aplicação realize a integração do código. Sendo assim, esta interface é implementada pela classe **LocalSimulation** (do pacote *simulation*). O polimorfismo consequente desta implementação permite à classe **ModelHandler** se relacionar com **LocalSimulation**, por meio da chamada do método **messageHandler**. Em outras palavras, a classe **ModelHandler** chamará o método **messageHandler**, a partir de um objeto do tipo **LocalMessageHandler**, que, na verdade, está na forma de um objeto **LocalSimulation**.

Da mesma forma que a execução no servidor, a chamada local cria um objeto da classe **ModelHandler** por meio de uma *thread*, porém, utilizando o segundo construtor. Neste caso, a instância é criada pela classe **LocalSimulation** passando os parâmetros *model* e *localMessageHandler* (seu tipo foi abreviado no diagrama como LMH por questões de espaço). O primeiro parâmetro refere-se aos dados do modelo, estruturados pela classe **ModelDatas** que está disponível no **CronoSim Framework**. O segundo parâmetro é uma referência da classe **LocalSimulation** sob a forma da interface implementada por ela.

Ao ser instanciada desta maneira, a classe **ModelHandler** irá considerar os

procedimentos correspondentes a uma simulação local. A *thread* será iniciada e o método **run** será chamado para dar início ao mecanismo de inicialização.

4.4.1.3 Mecanismo de Inicialização da Simulação

Ambos os modos de execução, explicados anteriormente, convergem ao mesmo mecanismo de inicialização da simulação. Ou seja, depois de estabelecer a comunicação entre a aplicação principal e o pacote *cronosimserver*, tanto localmente quanto em um servidor externo, o método **start** é chamado. Este método inicia os demais procedimentos, chamando o **createUserHome** para realizar as configurações necessárias, tais como: criar um identificador para este diretório e armazenar os dados do modelo recebido da aplicação principal (*model*).

Após a preparação do ambiente de controle, a simulação pode ser iniciada. Neste momento, o método **start** invoca duas *threads*, que são executadas pelas classes **SimulationRunner** e **SimulationObserver**, para execução e monitoramento da simulação, respectivamente. Antes do início da execução, a classe **SimulationRunner** verifica quantas máquinas serão necessárias, no caso de simulação distribuída. Isto é realizado pelo método **createMachinesFile** que escalona as máquinas de acordo com o número de processos existentes no modelo. Ou seja, se um modelo tem n processos, deve-se utilizar no máximo n máquinas.

Em sequência, são utilizados os recursos do **CronoSim Framework** para execução da simulação aplicando todos os dados do modelo e as configurações predefinidas. O comando correspondente a esta inicialização é realizado utilizando o MPJ-Express. Para isto, é adotada a classe **ProcessBuilder**, disponível na linguagem, para executar um comando no sistema.

Enquanto a simulação é executada, existe um mecanismo de monitoramento que permite acompanhar o andamento da simulação e também visualizar a distribuição de carga entre os nós do sistema distribuído. Neste caso, a *thread* responsável por este controle é executada pela classe **SimulationObserver**. Ao final da

simulação, os métodos responsáveis por enviar os resultados à aplicação principal são acionados e, por fim, a apresentação dos relatórios é tratada pela ferramenta, conforme foi explicado na seção sobre o ambiente gráfico.

4.5 Adaptações e Contribuições ao Framework

O **CronoSim Framework**, utilizado neste trabalho, fornece mecanismos que facilitam o desenvolvimento de simulações distribuídas. Os trabalhos realizados, anteriormente, a respeito do desenvolvimento deste *framework*, tiveram como principal objetivo, produzir recursos relacionados ao processamento distribuído, bem como os protocolos de sincronização. Como apresentado no capítulo 2, foram alcançadas diversas contribuições com as pesquisas relacionadas ao seu desenvolvimento, resultando nos trabalhos de Cruz (2009), Azevedo (2012) e Faria (2016).

Apesar de oferecer diversos recursos para desenvolver simulações em uma infraestrutura computacional distribuída, algumas características a respeito do modelo de simulação em si foram implementadas de forma estática, visando, apenas, validar o funcionamento do *framework*. Desta forma, os detalhes dos modelos não eram tratados nas versões anteriores, que possuíam o foco de testar o desempenho de simulações distribuídas e não obter os resultados do modelo.

Neste sentido, as versões anteriores do *framework* previam a simulação de sistemas discretos em geral, com intervalo de chegadas e tempo de atendimento que podem seguir diversas distribuições de probabilidades. Entretanto, embora exista a possibilidade de simular sistemas com estas características, os métodos responsáveis por tratar o modelo utilizavam a distribuição de probabilidade uniforme com intervalo fixo (1-10) para todos os elementos dos modelos.

A especificação do modelo considerava que a geração de entidades (entrada de novas entidades no sistema) só ocorre quando for executada a iteração correspondente ao atendimento realizado por um processo. Ou seja, ao processar uma entidade, é verificado se outra irá entrar no sistema e, em sequência, ser adicio-

nada na fila de eventos futuros do processo. No entanto, isto torna a entrada de entidades dependente do tempo de atendimento do processo.

Além disto, o mecanismo de execução da simulação foi implementado sem considerar a possibilidade de suspensão da execução, quando há necessidade, por exemplo, de analisar resultados parciais ou até mesmo finalizar a simulação antes do tempo determinado. Portanto, pode-se perceber que as funcionalidades inerentes à interação com o usuário podem ser aprimoradas a fim de atender a estas necessidades.

Além das dificuldades citadas nos parágrafos anteriores, existem, também, outras questões relacionadas à forma que o *framework* importava os dados do modelo. O método utilizado para receber os dados de um modelo era baseado em um arquivo de texto com uma estrutura predeterminada para representar as informações. Sendo assim, não existiam classes para representação de cada elemento do modelo.

Por conseguinte, deve-se levar em conta a implementação de mecanismos capazes de permitir a modelagem de sistemas, além de outros aspectos relacionados à interação com o usuário. CronoSim é a primeira ferramenta de simulação com ambiente de modelagem, execução e análise de resultados, que faz uso deste *framework* e, portanto, as questões relacionadas a estas funcionalidades puderam ser levantadas durante seu projeto. Desta forma, para oferecer flexibilidade na modelagem e maior controle sobre a simulação, este trabalho de mestrado contribui, também, no desenvolvimento dos novos mecanismos de modelagem que foram acoplados ao **CronoSim Framework**, possibilitando que outras aplicações também possam utilizá-las.

Neste contexto, foram desenvolvidas algumas classes para tratar os dados referentes ao modelo a ser simulado e, por conseguinte, adicionadas ao **CronoSim Framework**. O objetivo destas classes é fornecer as informações a respeito dos processos, permitindo especificar de forma estruturada o sistema e sua lógica de simulação. A nova abordagem permite representar os dados dos modelos e foi desenvolvida com base em um novo pacote, denominado "*model*", que será apre-

sentado na subseção 4.5.1. Algumas classes já existentes no *framework* também tiveram adaptações, a fim de atender às novas definições incorporadas ao projeto.

4.5.1 Pacote "Model"

O pacote *model* foi adicionado ao projeto do **CronoSim Framework**, possibilitando a geração correta dos dados durante a simulação de acordo com as características do sistema real. Outra contribuição importante é o tratamento dos dados resultantes da simulação, os quais não eram gerados na versão anterior do *framework*. A figura 31 apresenta o diagrama de classes do novo pacote adicionado ao **CronoSim Framework**, que contém as classes responsáveis por permitir a especificação do modelo.

Todas as classes disponíveis neste pacote implementam a interface **Serializable** (nativa do Java), o que possibilita salvar (em um arquivo externo) uma cópia completa de seu objeto e quaisquer outros objetos que ele faz referência, de modo que tal objeto pode ser recriado a partir da cópia "serializada". Com base nos novos recursos disponíveis, uma aplicação é capaz de utilizar uma estrutura orientada a objetos para descrição de um modelo. Isto é realizado por meio da classe **ModelDatas**, que contém o arcabouço para representar todos elementos do modelo e suas relações. Em vista disto, a especificação de um modelo utilizando o **CronoSim Framework** se torna simples, através do uso desta classe.

No centro do diagrama, pode-se observar a classe **TemplateNode**, que consiste em uma classe abstrata utilizada para representar os 3 componentes básicos de um modelo: entrada, processo e saída. Estes elementos possuem, em comum, os atributos: nome, identificador, contador de entidades (para indicar o número de entidades que transitaram por meio do elemento em questão durante a simulação) e os dados referentes aos valores estimados do tempo.

Os dados sobre as estimativas de tempo são abstraídos pela classe **TimeDatas**. Os atributos envolvidos referem-se aos elementos entrada e processo. A tabela 6

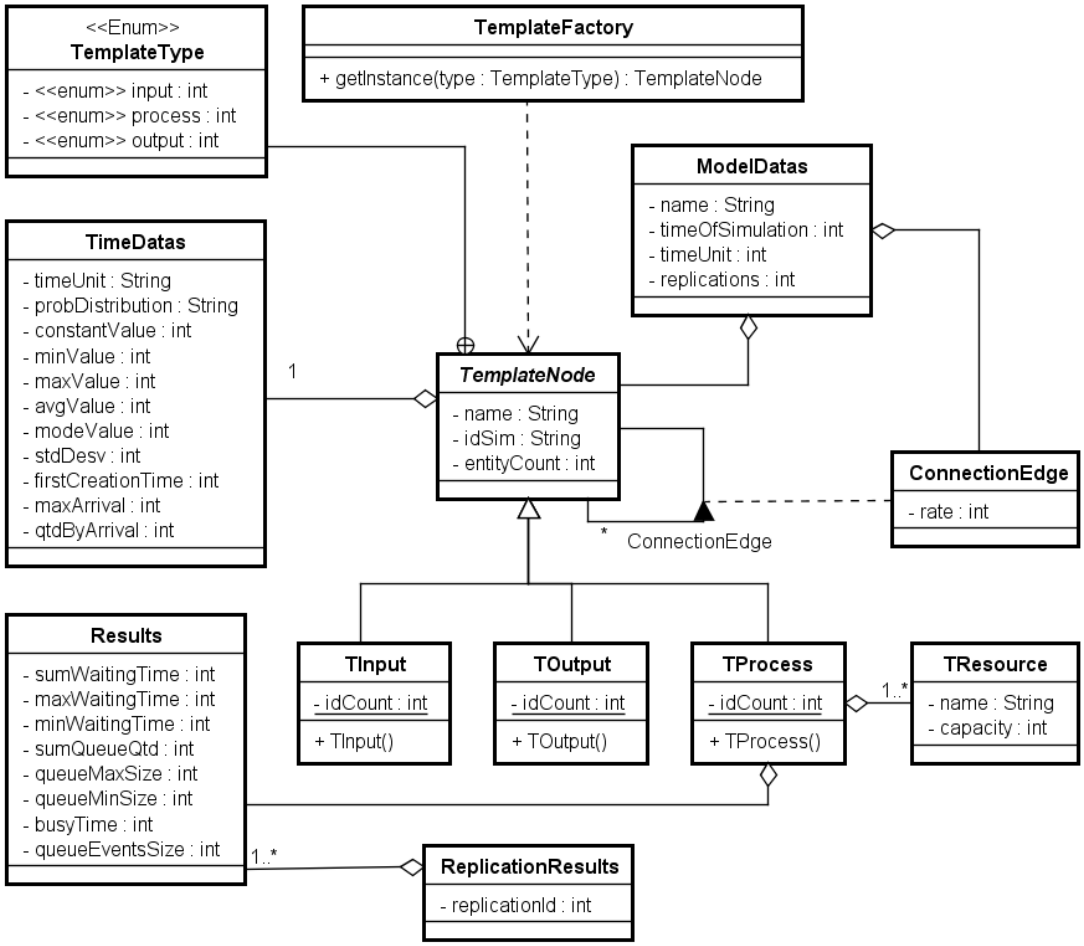

Figura 31: Diagrama de classes do novo pacote adicionado ao **CronoSim Framework**

mostra a descrição das informações que podem ser representadas nesta classe. Os valores associados podem ser definidos em: segundos, minutos, horas ou dias. Um modelo pode ser especificado com elementos baseados em diferentes unidades de tempo, ficando a cargo do programa fazer a conversão para uma unidade mínima durante a simulação.

As classes concretas **TInput**, **TProcess** e **TOutput** são especializações de **TemplateNode**, que representam os respectivos elementos do modelo. Cada uma

Tabela 6: Descrição das informações sobre as estimativas de tempo

Atributos	**Descrição**	**Elemento que Utiliza**
timeUnit	Unidade de tempo.	Entrada / Processo
probDistribution	Distribuição de probabilidade	Entrada / Processo
constantValue	Valor constante	Entrada / Processo
maxValue	Valor máximo	Entrada / Processo
minValue	Valor mínimo	Entrada / Processo
modeValue	Valor da moda	Entrada / Processo
avgValue	Valor médio	Entrada / Processo
stdValue	Desvio Padrão	Entrada / Processo
firstCreationTime	Tempo da primeira geração de eventos	Entrada
maxArrival	Máximo de chegadas. Ou seja, número máximo de entidades que podem ser geradas	Entrada
qtdByArrival	Quantidade de entidade por chegada	Entrada

contém um contador criado, a partir de um atributo estático, que é incrementado a cada vez que um objeto de sua classe é criado. Esse mecanismo é útil na construção de identificadores para os elementos adicionados durante a modelagem. Os construtores destas classes definem o tipo correspondente ao objeto. Para isto, a classe **TemplateNode** possui a declaração dos tipos de dados enumeráveis em **TemplateType**, relativos a cada elemento do modelo.

Para possibilitar a modelagem de sistemas levando em consideração a capacidade de atendimento, ou seja, o número de servidores (recursos) envolvidos em um processo, foi especificada a classe **TResources**. Ela representa um recurso que pode ser associado a um processo, contendo os atributos: nome e capacidade (indica o número de entidades que é possível atender a cada vez). Dessa forma, a classe **TProcess** contém uma lista de objetos do tipo **TResources**.

Para representar o fluxo das entidades entre os elementos, utiliza-se a classe **ConnectionEdge**. A partir dela é possível estabelecer todas as ligações entre os nós (elementos) do sistema e informar as probabilidades para cada alternativa, nos casos em que existam mais de uma opção para transição.

Sendo assim, uma aplicação pode especificar modelos utilizando os recursos

fornecidos neste novo pacote do *framework* através das classes correspondentes aos elementos. A especificação de modelos baseia-se, então, em instâncias de objetos dos tipos **TemplatesNodes** e **ConnectionEdge**. Por conseguinte, todas informações geradas podem ser atribuídas a um objeto da classe **ModelDatas**. Além dos detalhes relacionados aos elementos, os parâmetros da simulação também devem ser especificados, como nome (para identificação), o tempo de simulação e o número de replicações.

Afim de oferecer uma maneira simples para obter instâncias dos objetos concretos de **TemplateNode**, foi implementada a classe **TemplateFactory**, que fornece o método estático **getInstance**. A ideia consiste em criar um objeto correspondente a um dos 3 elementos, de acordo com o tipo especificado no parâmetro do método. Desta forma, as aplicações, para especificar os componentes do modelo, podem usar este mecanismo.

Por fim, as funcionalidades relativas ao tratamento dos resultados durante a simulação baseiam-se na classe **Results**, que, por sua vez, contém atributos referentes aos dados que serão utilizados para geração dos resultados. Cada objeto de **TProcess** possui uma instância de **Results** e, assim, cada processo atualiza os valores destes dados a cada ocorrência de um evento no processo.

Ao final da simulação, os resultados são gerados de acordo com os valores, atualizados, destes atributos. Para melhor compreensão da classe **Results**, a tabela 7 apresenta as informações abstraídas por ela. A atualização dos valores destes campos é realizada de acordo com o avanço do tempo de simulação, que depende dos parâmetros das probabilidades de ocorrência de um evento.

O tempo da simulação avança com base nos dados fornecidos ao modelo. Neste caso, a progressão do tempo consiste em um incremento, que, por sua vez, é gerado aleatoriamente a partir de uma distribuição de probabilidade a cada iteração. Os dados obtidos após este avanço correspondem ao tempo de atendimento (*delay time*) e ao tempo de espera (*waiting time*). Desta forma, a cada vez que uma entidade é atendida em um processo, estes dois dados são utilizados para atualizar

Tabela 7: Descrição dos dados utilizados para geração dos resultados da simulação

Atributo	**Descrição**
totalWaitingTime	Soma do tempo de espera de todas entidades
maxWaitingTime	Maior tempo de espera
minWaitingTime	Menor tempo de espera
sumQueueQtd	Soma da quantidade de entidades na fila a cada iteração
queueMaxSize	Tamanho máximo da fila
queueMinSize	Tamanho mínimo da fila
totalBusyTime	Tempo de ocupação total
queueEventsSize	Tamanho da fila ao final do tempo de simulação

os atributos da classe **Results**.

Neste contexto, torna-se possível obter as métricas requeridas em uma simulação, a partir dos valores atualizados de **Results**. Segundo Chwif e Medina (2014), estas métricas referem-se às medidas de desempenho que representam as respostas do sistema, por exemplo: peças produzidas por hora, tempo médio de espera em fila, taxa de utilização dos operadores, etc. A tabela 8 apresenta as equações aplicadas para geração destas medidas.

Tabela 8: Medidas de desempenho geradas na simulação

Medidas	**Valores utilizados para obtenção**
Entidades atendidas no processo	= entityCount (da classe **TemplateNode**)
Entidades não atendidas	= queueEventSize
Tamanho médio da fila	= sumQueueQtd / entityCount
Tempo médio de espera na fila	= totalWaitingTime / entityCount
Tempo mínimo de espera na fila	= minWaitingTime
Tempo máximo de esepra na fila	= maxWaitingTime
Taxa de utilização	= totalBusyTime / “Tempo da Simulação”
Tempo ocupado	= totalBusyTime
Tempo ocioso	= “Tempo da Simulação” - totalBusyTime

4.5.2 Adaptações Realizadas nos Demais Pacotes

As alterações realizadas nas classes implementadas nas versões anteriores do **CronoSim Framework** foram fundamentadas nos seguintes objetivos:

1. Melhorar a técnica utilizada para importar os dados dos modelos, que era realizada a partir de arquivos de textos pré-formatados.

2. Complementar o mecanismo de simulação, adicionando métodos para geração de números aleatórios, utilizando a distribuição de probabilidade escolhida pelo usuário.

3. Desenvolver métodos para geração das medidas de desempenho, que ainda não eram implementados nas versões anteriores.

4. Fornecer um meio de interação com a aplicação, para tornar possível o gerenciamento sobre a execução da simulação. Isto refere-se aos mecanismos de suspensão da simulação para obter dados parciais e realizar um monitoramento dos processos no sistema distribuído.

Neste contexto, as adaptações ao *framework* representam melhorias significativas em relação às funcionalidades relacionadas à interação com uma aplicação. Portanto, considera-se esta conjuntura, uma contribuição relevante deste trabalho de mestrado. Os parágrafos seguintes apresentarão, de maneira sucinta, as principais modificações e implementações atribuídas às novas características definidas.

A integração do pacote *model* com o restante do projeto ocorre de maneira simples, devido às características inerentes a Orientação a Objetos. Na nova abordagem, a classe **Process** passa a não existir, dando lugar a uma interface com o mesmo nome, como apresentado na figura 32. Os atributos e métodos das classes do pacote *simulation* não foram exibidos no diagrama, mas pode-se notar que a utilização de interface permitiu a exploração do polimorfismo. Com isto, as classes **SimulationProcess** e **ControllerProcess** implementam esta nova interface. Além disso, **SimulationProcess** torna-se uma especialização de **TemplateNode**. Deste modo, os objetos de **SimulationProcess** herdam todos os detalhes implementados na superclasse, como identificador, contador de entidades, tipo de elemento, além de todos os dados sobre o tempo.

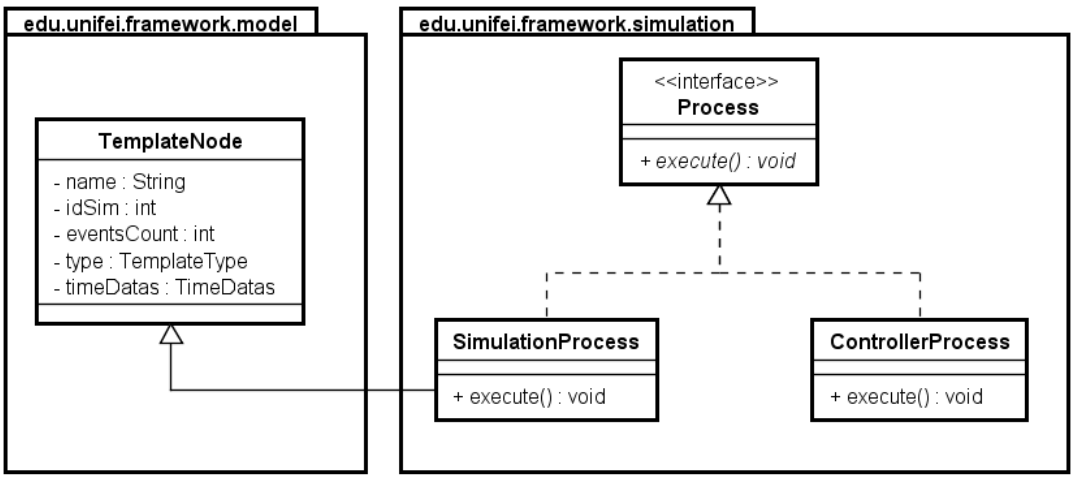

Figura 32: Integração do pacote *model* no projeto do **CronoSim Framework**

A interface **Process** fornece uma abstração para um processo alocado na execução computacional realizada durante a simulação. Sendo assim, um objeto deste tipo representa um processo real executando uma tarefa, seja envolvido na simulação propriamente dita ou relacionado ao controle.

Portanto, com estas alterações foi possível atender os três primeiros objetivos citados anteriormente. Todavia, o desenvolvimento de um mecanismo para gerenciamento da simulação foi baseado em arquivos de controle, da mesma forma utilizada no componente **CronoSim Server**, como já discutido.

4.6 Considerações Finais

A ferramenta CronoSim que foi apresentada neste capítulo foi desenvolvida para prover recursos que possibilitam a execução de simulações distribuídas. O ambiente gráfico foi projetado para atender às fases da simulação correspondentes à modelagem, execução e análise de resultados. Os mecanismos utilizados no projeto da ferramenta fazem parte de um conjunto de pesquisas que vêm sendo realizado pelo Grupo de Pesquisas em Engenharia de Sistemas e de Computação

(GPESC) e que contribuíram diretamente para o desenvolvimento deste trabalho. No entanto, conforme apresentado neste capítulo, diversos outros mecanismos foram implementados, a fim de alcançar o objetivo proposto neste trabalho de mestrado.

A ferramenta CronoSim será disponibilizada à comunidade científica. O público alvo será usuários que pretendem desenvolver simulações de eventos discretos e aproveitar o poder computacional de sistemas distribuídos para melhorar o desempenho da execução. Além disto, será disponibilizado o código fonte com o projeto completo, com o objetivo de permitir que outros pesquisadores e desenvolvedores possam contribuir com melhorias, realizar adaptações ou qualquer modificação. Desta forma, pode ser vista, também, como um ambiente de testes para pesquisadores na área de Simulação Distribuída.

5 Resultados Experimentais

Este capítulo apresenta e discute os resultados experimentais alcançados com o uso da ferramenta CronoSim. Os resultados serão descritos em duas partes distintas. A primeira parte demonstra a usabilidade da ferramenta, utilizando modelos tradicionais para validar os métodos que geram os resultados da simulação e, também, o seu funcionamento. A segunda parte é um estudo sobre a melhoria de desempenho que a ferramenta oferece em virtude da paralelização.

5.1 Primeira Parte: Validação dos Métodos que Geram os Resultados da Simulação

Nesta seção, será descrito o funcionamento da ferramenta CronoSim, mostrando a criação de um modelo simples com o propósito de demonstrar a utilização da ferramenta e apresentar as características sobre a usabilidade. Com a simulação deste modelo, foi possível demonstrar que esta versão de CronoSim apresenta resultados compatíveis com Arena, que é uma das mais populares ferramentas de uso comercial (KELTON; SADOWSKI; STURROCK, 2014).

O cenário que será apresentado consiste em um exemplo que foi extraído de Parreira Jr (2010). Vale destacar que esta avaliação considera que a coleta e análise dos dados de entrada do sistema já foram realizadas. Deste modo, o foco será a especificação do modelo utilizando os dados previamente definidos. Portanto, tem-se disponível os valores correspondentes às estimativas do intervalo entre chegadas

e do tempo gasto para descarregar os navios.

Neste contexto, este exemplo consiste em um porto onde navios chegam (ininterruptamente) a intervalos baseados em uma distribuição exponencial de 8 horas e o tempo gasto para descarregar baseia-se na distribuição triangular de 3, 5 e 10 horas. Desta forma, pretende-se simular o período de 1 ano de atividade deste porto, de modo que seja possível responder as seguintes perguntas:

1. Qual a quantidade de navios que descarregaram no porto?
2. Qual a taxa de ocupação do porto?
3. Qual a quantidade máxima de navios no porto?
4. Qual a quantidade média de navios no porto?
5. Qual o tempo médio de espera na fila?
6. Qual o tempo máximo de espera na fila?

Para melhor compreensão dos recursos oferecidos pela ferramenta CronoSim, serão descritos todos os passos para modelar este sistema simples até obter os resultados. Em primeiro lugar, é preciso identificar os elementos do modelo para transcrevê-lo ao ambiente de modelagem, ou seja, apontar quais são as entidades e atividades envolvidas. Neste caso, os navios são as entidades que chegam ao sistema e a atividade inerente ao processo é o descarregamento da carga dos navios. Quando um navio termina o descarregamento, deve sair do sistema. Portanto, é possível identificar três itens para serem adicionados ao modelo:

1. um bloco de **entrada**, para representar a chegada dos navios ao porto e especificar a função que define o intervalo de chegadas;
2. um bloco de **processo**, para representar o descarregamento dos navios no porto;

3. um bloco de **saída**, para representar a saída do porto.

A figura 33 ilustra a tela onde os componentes foram adicionados e a interface para especificação dos dados do bloco de entrada. O nome "Chegada de Navios" foi atribuído ao bloco e o tipo de entidade que entra no sistema é "Navio". O valor correspondente ao tempo entre chegadas foi definido conforme os dados descritos neste exemplo.

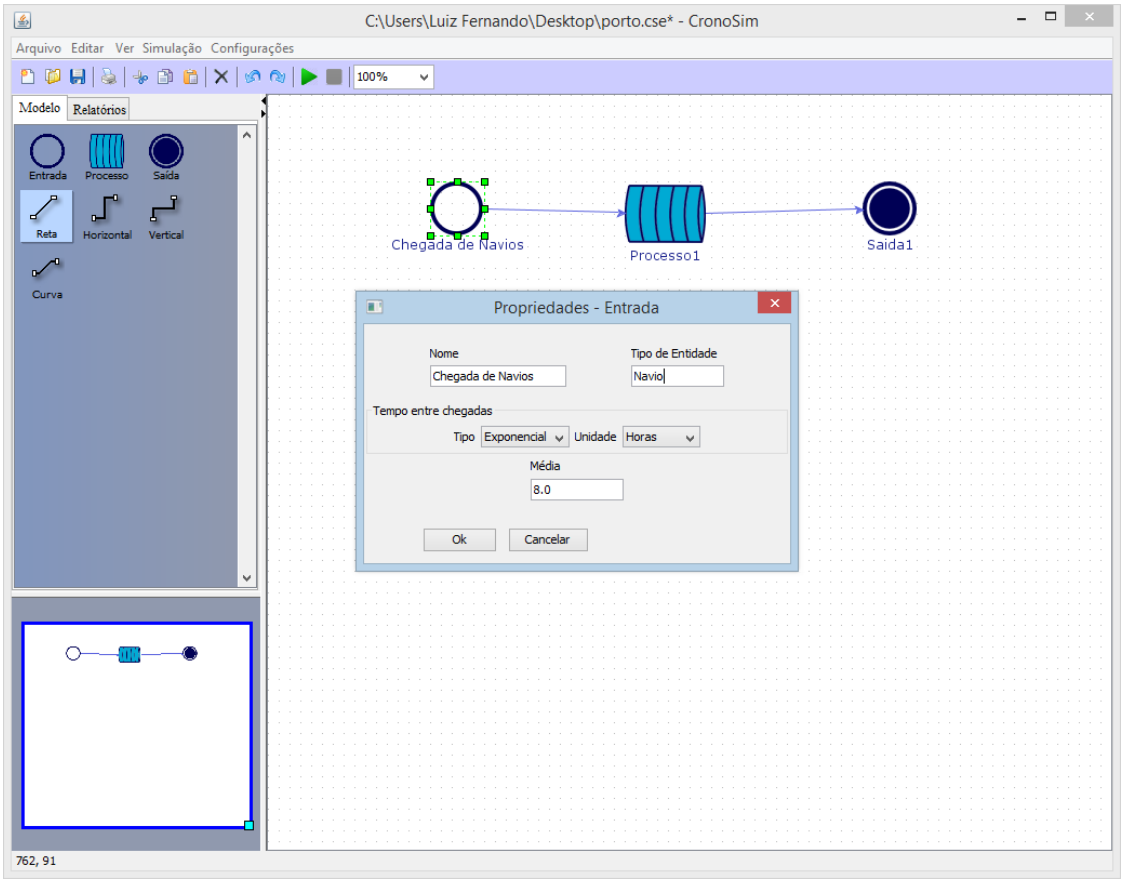

Figura 33: Iniciando a modelagem de um sistema simples

Da mesma forma, deve-se especificar as informações referentes ao processo (descarregamento do navio no porto). A figura 34 mostra os dados definidos para este processo do modelo. O nome "Porto" foi atribuído a este processo e os valores probabilísticos do tempo de descarregamento foram definidos. Considerou-se, neste

exemplo, que os navios são descarregados utilizando um guindaste e, por isto, foi adicionado este recurso ao processo. Por fim, pode-se definir um nome para o bloco de saída como, por exemplo, “Saída de Navios”.

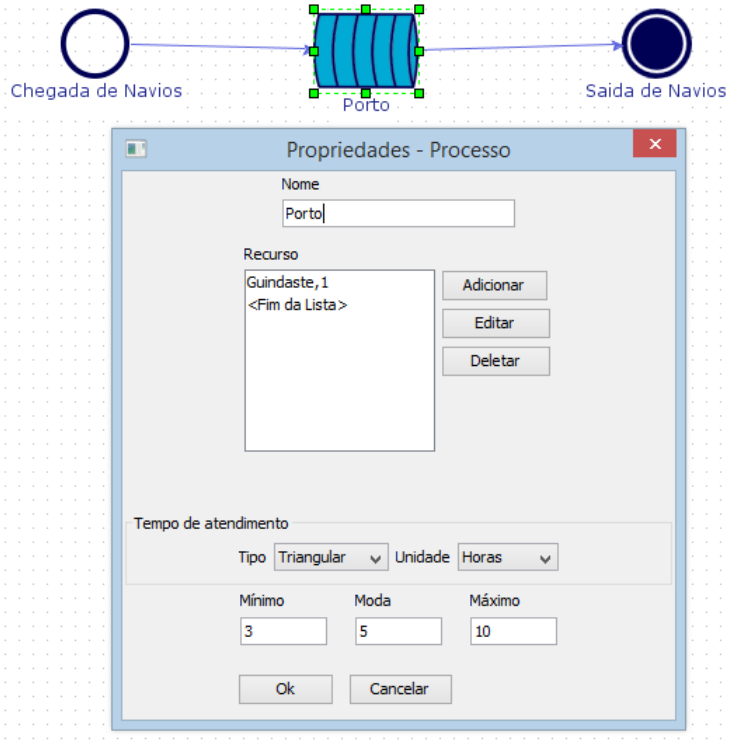

Figura 34: Especificação dos dados de um processo do modelo

Após realizar a especificação do modelo, a próxima etapa consiste em definir os parâmetros da simulação, como o número de replicações, duração total da simulação, duração, por dia, de funcionamento do sistema e a unidade de tempo base para exibição nos relatórios. A figura 35 mostra a tela com os valores já configurados para este exemplo. Definiu-se que serão realizadas 10 replicações e a duração da simulação é de 365 dias, correspondentes a 1 ano. Como os navios chegam ininterruptamente, foi definido que o sistema funciona 24 horas por dia. Para facilitar a compreensão dos relatórios, definiu-se que os valores serão exibidos utilizando “horas” como unidade de tempo base.

Os resultados para as simulações utilizando as ferramentas Arena e CronoSim

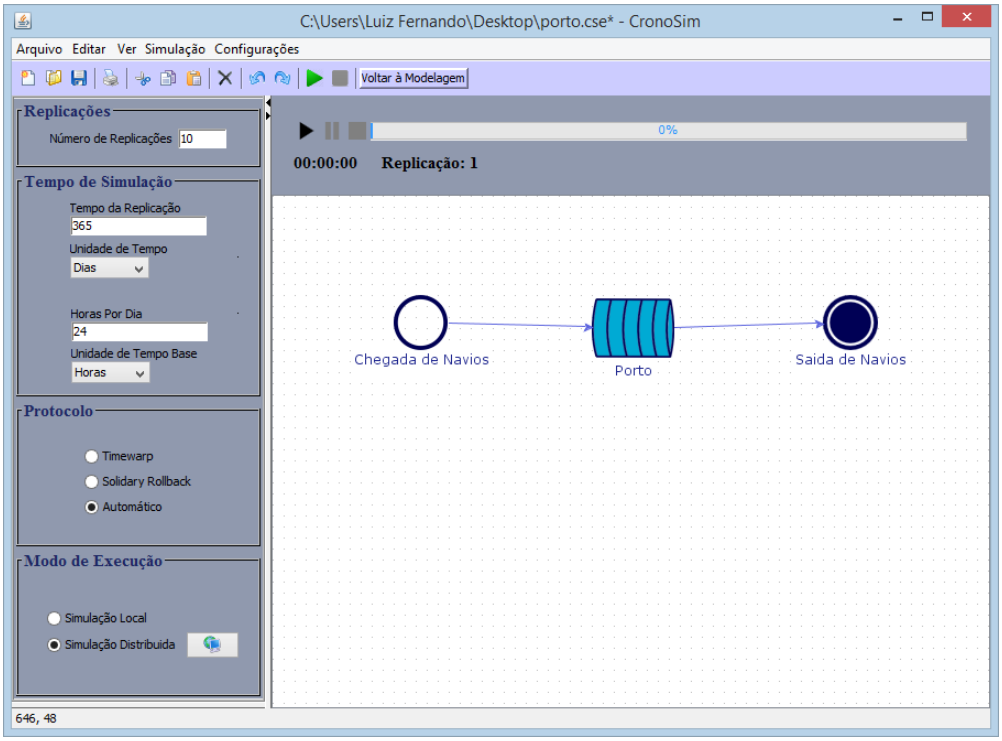

Figura 35: Especificação dos parâmetros da simulação

são apresentados nas tabelas 9 e 10, respectivamente. A tabela 11 sintetiza os resultados da simulação através da ferramenta CronoSim e, também, mostra os resultados do mesmo modelo, considerando o uso da ferramenta Arena. Com estes valores é possível realizar uma comparação, verificando se os resultados estão numericamente semelhantes a ferramenta Arena. Como pode-se notar, os resultados obtidos na simulação por meio da ferramenta CronoSim são coerentes. O intervalo de confiança de 90% destes resultados são apresentados na tabela 12.

Tabela 9: Resultados para a simulação usando a ferramenta Arena

	Replicações									
	1	2	3	4	5	6	7	8	9	10
1- Nº descarregados	1148	1102	1118	1125	1153	1175	1088	1075	1080	1084
2- Taxa de ocupação	78,65	76,23	76,48	75,34	78,79	79,56	75,48	74,16	73,22	74,29
3- Nº máx. de navios	10	9	12	10	10	8	7	9	10	13
4- Nº médio de navios	1,36	1,16	1,22	1,02	1,41	1,44	1,04	0,92	1,13	1,37
5- Tempo méd. de espera	10,38	9,23	9,52	7,97	10,70	10,72	8,33	7,45	9,13	11,04
6- Tempo máx. de espera	59,58	48,57	71,26	55,40	53,02	47,45	40,86	50,23	55,73	85,92

Tabela 10: Resultados para a simulação usando a ferramenta CronoSim

	Replicações									
	1	2	3	4	5	6	7	8	9	10
1- Nº descarregados	1159	1121	1168	1080	1092	1149	1078	1132	1075	1110
2- Taxa de ocupação	78,95	75,84	79,11	74,12	75,5	78,24	73,89	76,45	72,86	76,12
3- Nº máx. de navios	10	12	8	9	13	10	7	10	12	10
4- Nº médio de navios	1,13	1,10	1,44	0,90	1,36	1,41	1,01	1,13	1,23	1,32
5- Tempo méd. de espera	9,56	9,22	10,74	7,91	11,05	10,75	8,98	9,32	9,61	10,98
6- Tempo máx. de espera	69,32	72,15	51,14	49,98	81,12	51,11	41,37	57,17	52,44	79,32

Tabela 11: Comparação dos resultados das ferramentas Arena e CronoSim

	Arena		CronoSim	
	Média	Desvio Padrão	Média	Desvio Padrão
Quantidade de navios que descarregaram	1114,8	34,84	1116,4	34,89
Taxa de ocupação do porto	76,22	2,16	76,11	2,15
Quantidade máxima de navios no porto	9,8	1,75	10,1	1,85
Quantidade média de navios no porto	1,21	0,18	1,2	0,1796
Tempo médio de espera na fila	9,45	1,26	9,8131	1,0358
Tempo máximo de espera na fila	56,8016	13,05894	56,910	11,839

Tabela 12: Intervalos de confiança das ferramentas Arena e CronoSim

	Intervalo de Confiança de 90%			
	Arena		CronoSim	
Quantidade de navios que descarregaram	1096,62	1132,98	1098,20	1134,60
Taxa de ocupação do porto	75,09	77,35	74,99	77,23
Quantidade máxima de navios no porto	8,89	10,71	9,13	11,07
Quantidade média de navios no porto	1,11	1,30	1,11	1,30
Tempo médio de espera na fila	8,79	10,10	9,27	10,35
Tempo máximo de espera na fila	49,99	63,62	50,73	63,09

5.2 Segunda Parte: Análise de Desempenho

A análise realizada nesta seção enfatiza o desempenho alcançado pela simulação em uma arquitetura distribuída de computadores. Os resultados correspondentes a esta análise foram apresentados e publicados nos Anais do XLVII Simpósio Brasileiro de Pesquisa Operacional (NUNES et al., 2015). Além disso, parte destes resultados estão presentes no trabalho de Faria (2016), que é coautor do artigo publicado.

Os experimentos utilizando o CronoSim foram executados em um *cluster* customizado do GPESC (Grupo de Pesquisas em Engenharia de Sistemas e de Com-

putação), da Universidade Federal de Itajubá, composto por cinco máquinas Core 2 Duo CPU Q6600 @ 2.40GHz (L2, 4Mb), 4Gb DDR III, Rede Ethernet 10/100 com sistema operacional Linux. O objetivo foi estabelecer um comparativo de desempenho entre simulações executadas em um ambiente centralizado e em um sistema distribuído. Sendo assim, foram realizados simulações em 1 computador e, também, utilizando arranjos com 2, 3, 4 e 5 computadores do *cluster*.

5.2.1 Modelos Utilizados

Os testes foram realizados com modelos de 4, 50 e 100 processos lógicos, construídos pelo CronoSim, de modo a calcular o tempo gasto para executar 10.000 eventos em cada modelo. Nos três modelos criados para os testes, todos os processos tratavam de chegadas e atendimentos marcovianos com um único centro de serviço (Notação *Kendall-Lee* M/M/1) (CHWIF; MEDINA, 2014). Além disto, os intervalos de chegadas e tempo de atendimento foram estabelecidos com distribuição uniforme entre 1 a 10.

Cada modelo foi simulado de modo a calcular os respectivos tempos médios de execução e desvios-padrões ao longo de 10 replicações para cada experimento. Entretanto, o modelo de 4 processos não necessitou de testes com 5 processadores, pois utilizaria, no máximo, 4 destes.

As figuras 36 e 37 mostram, respectivamente os modelos de 4 e 50 processos lógicos. Por questões de espaço, não será ilustrado o modelo de 100 processos que contém organização semelhante ao modelo de 50 processos, porém com as seguintes características:

- Possui 10 entradas com 5 fluxos de transição para outros processos. Cada fluxo possui mesma probabilidade de ocorrência;
- Cada processo, possui 2 fluxos de transição para outros processos. Cada fluxo possui 50% de probabilidade de ocorrência.

- Os processos finais na série, possuem 100% de probabilidade de não gerar novos eventos, portanto possuem apenas um fluxo de saída.

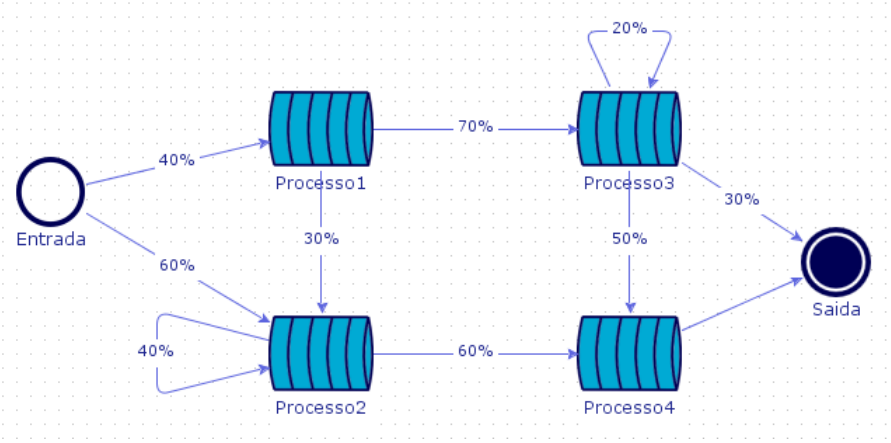

Figura 36: Modelo com 4 processos

5.2.2 Análise e Discussão

A tabela 13 apresenta os resultados destes experimentos. Os números correspondem ao tempo real de execução e está descrito em segundos.

Os critérios adotados foram: o Tempo Médio da Simulação (TMS), que representa a média do tempo que cada processo levou para completar o processamento; o Tempo da Simulação (TSIM), que indica a quantidade de tempo que a simulação levou para executar, isto é, o tempo do processo que mais demorou e, por fim, os respectivos Desvios Padrões (DP) do TMS e do TSIM.

O modelo de 4 processos, apesar de simples, foi criado para demonstrar o comportamento da simulação de um sistema com pouca complexidade. Conforme os resultados apresentados na tabela 13, pode-se notar que o ganho de desempenho deste modelo não foi significativo. Isto se deve às características do modelo, que possui processos em série, criando uma dependência na comunicação entre os respectivos processos lógicos. A tabela 14 apresenta os intervalos de confiança de 95% das métricas TMS e TSIM.

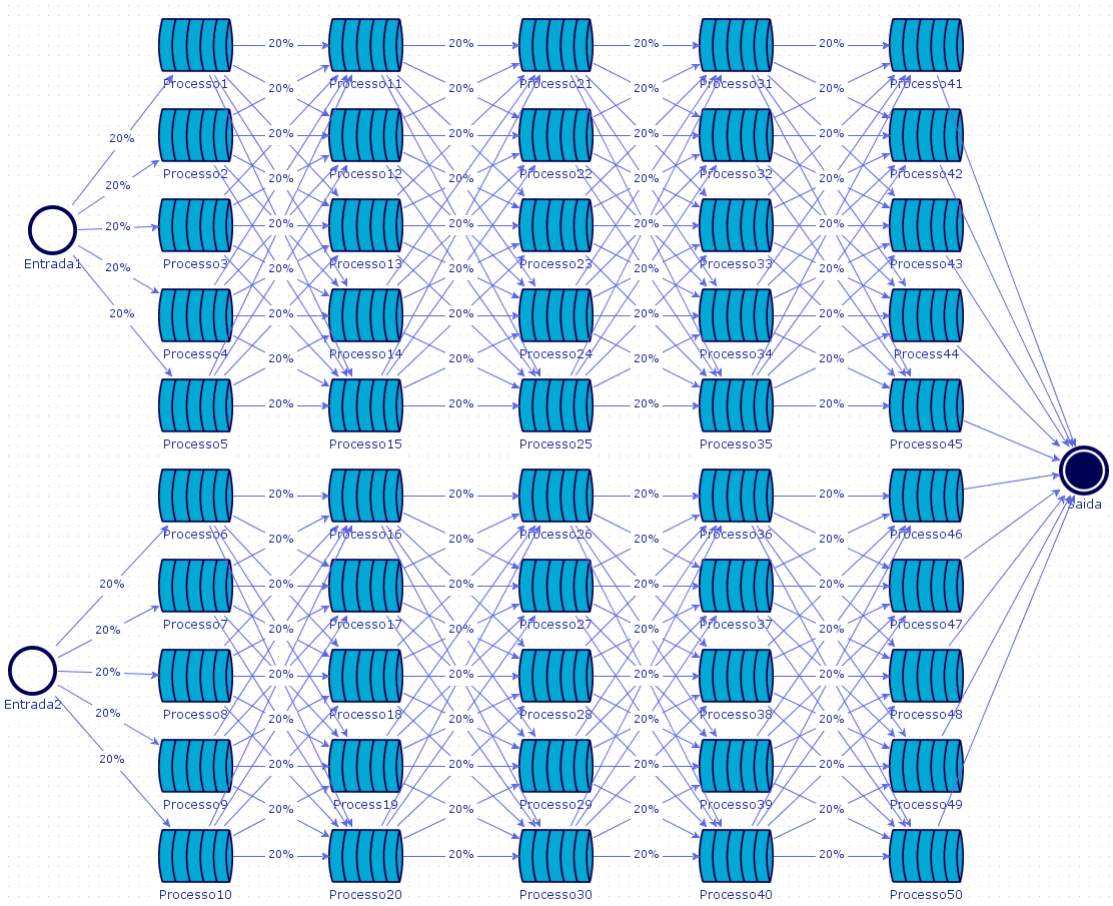

Figura 37: Modelo com 50 processos

Naturalmente, apesar da possibilidade de simular modelos com poucos componentes, as vantagens do CronoSim aparecem na simulação de modelos maiores. Para ilustrar este cenário, foram criados dois modelos com 50 e 100 centros de serviços respectivamente. Como explicado anteriormente, cada centro de serviço foi tratado como um processo lógico do sistema distribuído. Os modelos foram criados respeitando-se as taxas de nascimento e de serviço do modelo M/M/1 e distribuindo-se, aleatoriamente, as conexões entre os elementos. Apesar de possuir o dobro de processos, o modelo com 100 centros de serviços foi gerado com uma quantidade significativamente menor de ligações entre os elementos, cerca de 50% do total de ligações do modelo com 50 processos. Esta modificação mostra o

impacto da comunicação, mas ilustra o potencial de paralelismo da ferramenta.

Tabela 13: Comparativo de desempenho da simulação distribuída em diferentes números de máquinas

Modelo	Nº de nós	TMS	DP (TMS)	TSIM	DP (TSIM)
4 processos	1	100,50	0,71	184,00	1,41
	2	97,50	0,71	179,75	0,96
	3	97,75	0,96	179,00	0,82
	4	96,83	0,75	179,00	0,89
	5	-	-	-	-
50 processos	1	229,50	4,95	1986,50	47,38
	2	190,67	4,18	1505,14	47,15
	3	193,50	5,69	1690,10	65,19
	4	190,00	5,16	1533,60	58,32
	5	186,50	5,32	1411,50	56,36
100 processos	1	89,00	2,83	411,50	44,55
	2	75,83	1,94	364,50	32,27
	3	75,33	2,16	390,00	41,63
	4	73,50	3,87	395,00	50,91
	5	74,00	1,41	369,50	30,41

Tabela 14: Intervalos de confiança das métricas: TMS e TSIM

Intervalo de confiança de 95%					
Modelo	Nº de nós	TMS		TSIM	
4 processos	1	100,06	100,94	183,12	184,88
	2	97,06	97,94	179,16	180,34
	3	97,16	98,34	178,49	179,51
	4	96,37	97,30	178,45	179,55
	5				
50 processos	1	226,43	232,57	1957,14	2015,86
	2	188,08	193,26	1475,92	1534,36
	3	189,98	197,02	1649,70	1730,50
	4	186,80	193,20	1497,45	1569,75
	5	183,20	189,80	1376,57	1446,43
100 processos	1	87,25	90,75	383,89	439,11
	2	74,63	77,04	344,50	384,50
	3	73,99	76,67	364,20	415,80
	4	71,10	75,90	363,44	426,56
	5	73,12	74,88	350,65	388,35

Os indicadores de desempenho foram baseados nas seguintes métricas: *speedup* e eficiência. A primeira métrica é definida como a divisão do tempo gasto na simulação no ambiente sequencial (T_{seq}) pelo tempo gasto pela simulação paralela (T_{par}), enquanto a segunda é a divisão do *speedup* pelo número de processadores (TANG; LEE; HE, 2012). As equações 5.1 e 5.2 representam as definições para *speedup* e eficiência, respectivamente. O resultado ideal do *speedup* seria que seu valor aumentasse na mesma proporção do aumento do número de processadores, sendo que o valor ideal da eficiência deve ser 100% (NUNES et al., 2015).

$$\text{Speedup} = \frac{T_{seq}}{T_{par}} \tag{5.1}$$

$$\text{Eficiência} = \frac{\text{Speedup}}{\text{nº de processadores}} \tag{5.2}$$

A tabela 15 mostra o *speedup* e a eficiência das execuções paralelas em relação ao experimento centralizado para avaliar o quanto a simulação distribuída traz ganhos em termos de desempenho para a simulação.

Para melhor interpretação dos resultados discutidos neste capítulo, os gráficos das figuras 38, 39, 40 e 41 ilustram os dados apresentados na tabela 15, referentes aos indicadores de desempenho, *speedup* e eficiência, de cada teste. Os gráficos mostram as medidas, separadamente, para Tempo Médio da Simulação (TMS) e Tempo da Simulação (TSIM).

O cálculo do *speedup* e da eficiência das configurações paralelas, em relação à execução centralizada em um único computador, demonstra que por mais que a computação distribuída contribua com certa melhoria de desempenho na obtenção dos resultados da simulação, o *overhead* de mensagens sempre se faz presente, levantando questões como: balanceamento de carga e escalonamento de processos (NUNES et al., 2015).

Tabela 15: *Speedup* e eficiência sobre a simulação

Modelos	Número de processadores	*TMS*		*TSIM*	
		Speedup	Eficiência	*Speedup*	Eficiência
4 processos	2	1,03	51,5%	1,02	51,2%
	3	1,03	34,3%	1,03	34,3%
	4	1,04	25,9%	1,03	25,7%
	5	-	-	-	-
50 processos	2	1,20	66,0%	1,32	66,0%
	3	1,19	39,2%	1,18	39,2%
	4	1,21	32,4%	1,30	32,4%
	5	1,23	28,1%	1,41	28,1%
100 processos	2	1,18	58,9%	1,13	56,4%
	3	1,19	39,8%	1,06	35,2%
	4	1,21	30,3%	1,04	26,0%
	5	1,21	24,2%	1,11	22,3%

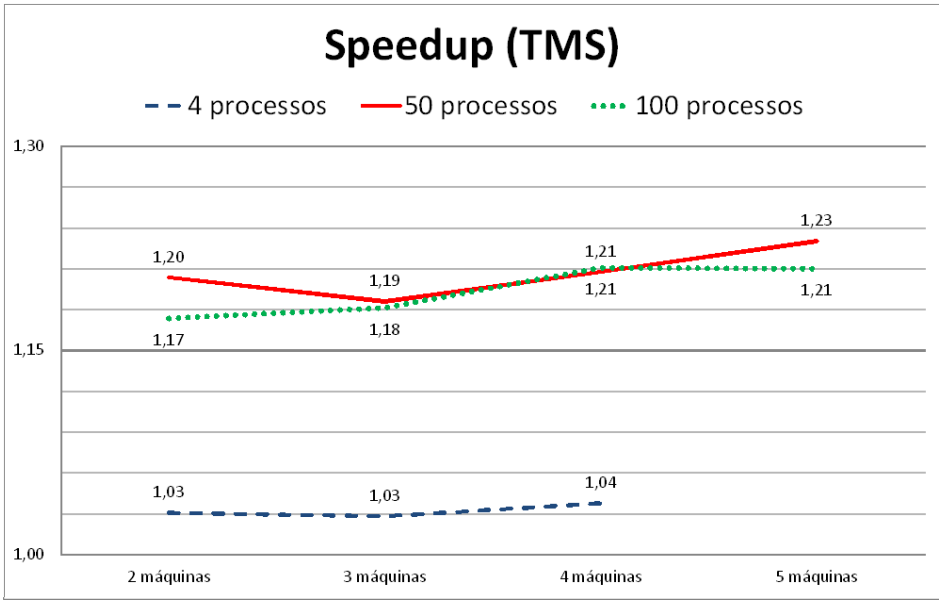

Figura 38: Gráfico dos resultados do *speedup* (TMS)

5.3 Considerações Finais

Neste capítulo, foram discutidos os resultados experimentais obtidos a partir da ferramenta Cronosim, desenvolvida neste trabalho de mestrado. Os experi-

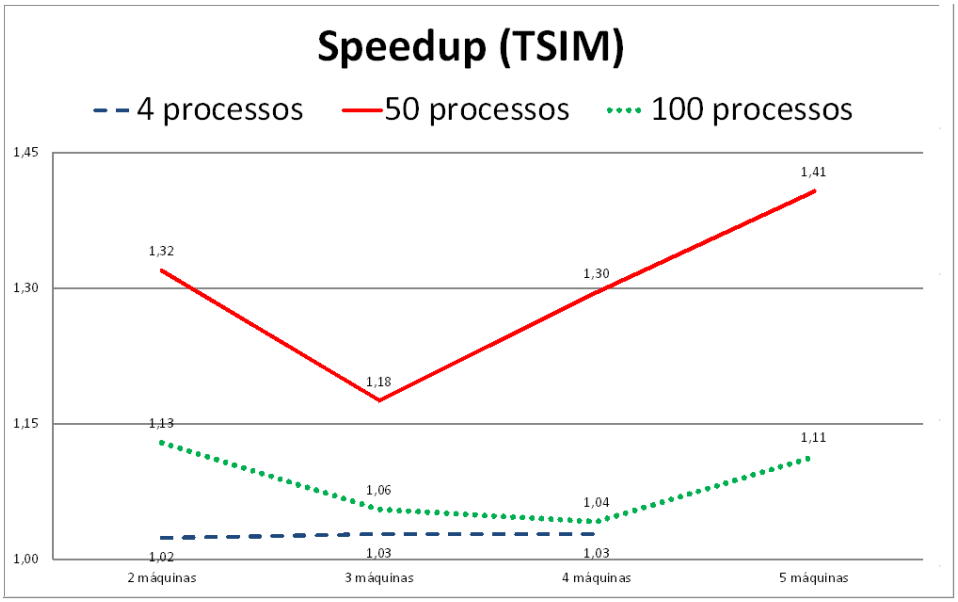

Figura 39: Gráfico dos resultados do *speedup* (TSIM)

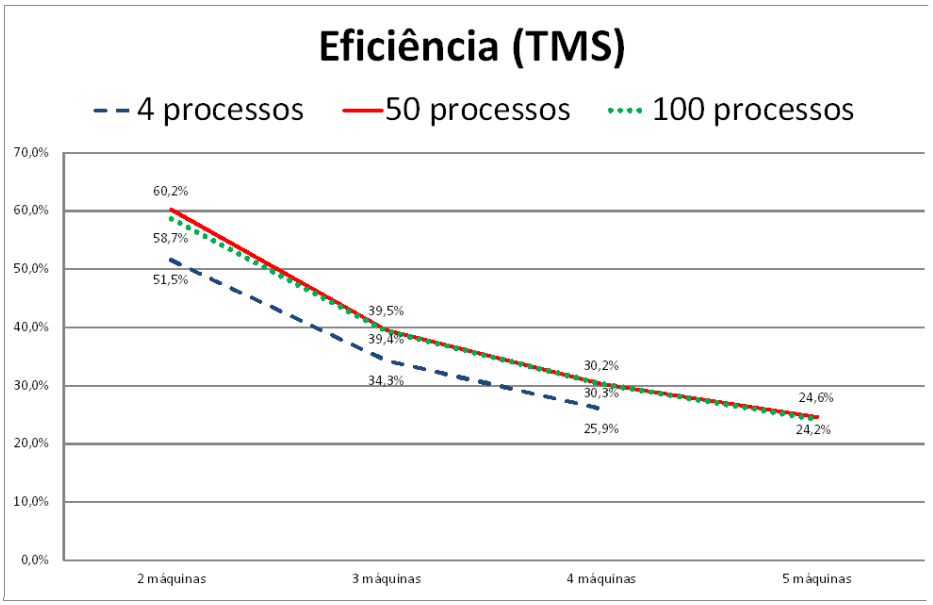

Figura 40: Gráfico dos resultados da eficiência (TMS)

mentos foram fundamentados em duas etapas distintas. A primeira consistiu na validação dos mecanismos de geração das variáveis estimadas, que foram utilizadas para representar os resultados da simulação. Foi possível demonstrar que os

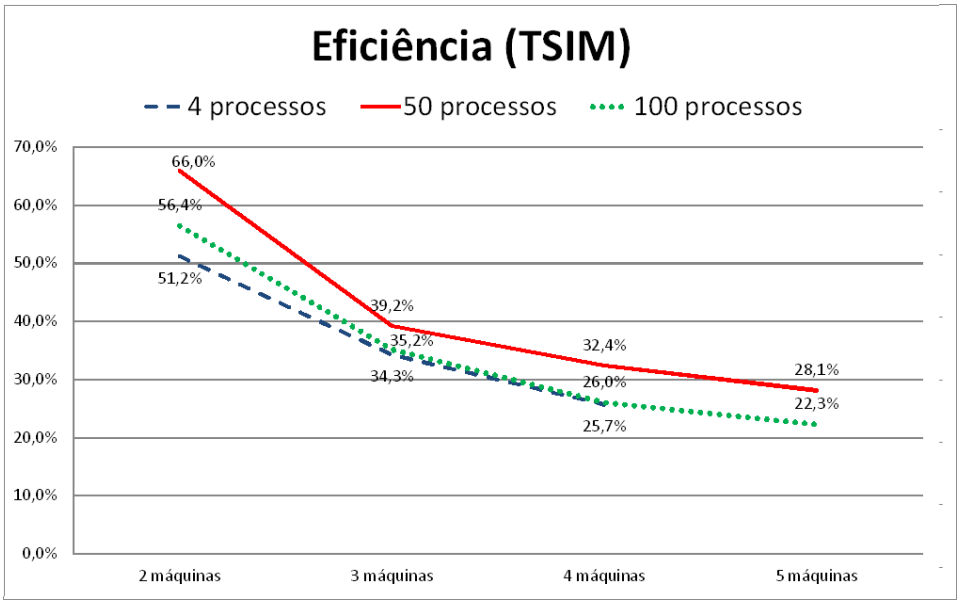

Figura 41: Gráfico dos resultados da eficiência (TSIM)

dados resultantes da simulação são coerentes com uma tradicional ferramenta de Simulação de Eventos Discretos.

A segunda etapa dos experimentos foi a análise de desempenho computacional do programa de simulação distribuída. As métricas utilizadas para calcular os indicadores de desempenho foram baseadas no tempo de execução dos processos das simulações. Os resultados demonstraram que essas métricas não atingem seus respectivos valores ideais, por conta do *overhead* de comunicação entre as máquinas, o que afeta diretamente o cálculo da eficiência. Mesmo assim, a diminuição do tempo total da simulação de modelos grandes é significativa, o que permite obter melhor desempenho que um programa de simulação centralizado em uma única máquina.

Com estes resultados, pode-se concluir que o impacto causado pelo *overhead* de comunicação sobre o desempenho, ainda, é um dos pontos críticos na abordagem distribuída, pois afeta, diretamente, o tempo de execução. Entretanto, conforme já discutido no capítulo 2, outras pesquisas encontradas na literatura vêm apresentando novos mecanismos, relacionados aos protocolos de sincronização e à migração

de processos, a fim de reduzir este impacto. Por sua vez, a ferramenta Cronosim permite que outras implementações de protocolos sejam adicionadas ao projeto, possibilitando a análise sobre o desempenho de outros mecanismos de sincronização que vêm sendo apresentados na literatura.

6 Conclusão

A ferramenta CronoSim, desenvolvida neste trabalho de mestrado, permite realizar simulações de eventos discretos em uma infraestrutura computacional distribuída. Esta ferramenta oferece aos usuários diversas características que têm como objetivo principal facilitar a elaboração de simulação distribuída. A paralelização da execução reduz o tempo de processamento e, assim, aumenta o desempenho.

Atualmente, diversos estudos vêm sendo realizados, a fim de melhorar o desempenho em Simulações de Eventos Discretos, sendo um fator relevante para os usuários. Com o desenvolvimento das tecnologias nesta área e surgimento de novos paradigmas computacionais, como "Computação em *Cluster*" e "Computação em Nuvem", o conceito de simulação paralela e distribuída tem amadurecido, mas ainda está em evolução. Novas plataformas de computação, tais como máquinas massivamente paralelas, GPUs (*Graphics Processing Unit*) e, inclusive, computação em nuvem estão fazendo mudanças transformadoras para a área de Computação como um todo.

Uma das principais dificuldades encontradas durante o desenvolvimento deste trabalho de mestrado foi a integração de outros projetos desenvolvidos pelos pesquisadores do Grupo de Pesquisas em Engenharia de Sistemas e de Computação (GPESC), uma vez que o resultado obtido neste trabalho foi baseado na composição destes projetos. Esta etapa demandou certo tempo, devido à complexidade dos conceitos envolvidos no tratamento dos mecanismos inerentes à simulação distribuída, tais como, a comunicação e a sincronização dos processos escalonados em

cada nó do sistema computacional distribuído.

Além disso, a interação entre o módulo **CronoSim Framework** e **CronoSim Application** exigiu um tratamento especial, pois o *framework* registrava os dados para testar o desempenho de simulações distribuídas, mas não dados referentes ao modelo simulado. Os métodos de tratamento dos dados do modelo utilizavam valores estáticos e não valores estipulados pela modelagem do usuário. Isto acontecia devido ao fato deste *framework* ter sido utilizado nos trabalhos anteriores, apenas para demonstrar o comportamento dos protocolos de sincronização e, também, as melhorias de desempenho alcançadas por meio da paralelização.

Deste modo, o projeto da ferramenta CronoSim levou em conta a implementação de mecanismos no *framework* capazes de permitir a modelagem de sistemas, além de outros aspectos relacionados à interação com o usuário. Diversas questões relacionadas a estas funcionalidades puderam ser levantadas, o que permitiu desenvolver novos mecanismos de modelagem que foram acoplados ao **CronoSim Framework**, possibilitando, inclusive, que outras aplicações possam utilizá-los.

Esta flexibilidade relacionada à reutilização de código é uma vantagem relevante alcançada neste trabalho, possibilitando o uso dos componentes por outras aplicações. Como foi apresentado, a ferramenta CronoSim possui uma arquitetura composta por diversos pacotes com funcionalidades específicas e componentes distintos. Por sua vez, o componente **CronoSim Server** fornece recursos para permitir o gerenciamento de uma simulação e, também, estabelecer a conexão entre uma aplicação e uma rede de computadores. Este componente funciona como uma biblioteca que pode ser inserida em outros projetos para tratar a execução utilizando o *framework* em questão.

6.1 Contribuições deste Trabalho

A principal contribuição deste trabalho foi o projeto e implementação de uma ferramenta para simulação de eventos discretos, que pode ser executada de forma

distribuída em um *cluster*. Esta ferramenta oferece um ambiente gráfico que foi projetado para possibilitar o desenvolvimento de simulações, abrangendo as fases: modelagem, execução e análise de resultados. A elaboração deste projeto foi fundamentada no oferecimento de recursos que permitem obter melhorias no desempenho computacional em virtude da paralelização.

Além disso, novas pesquisas na área de Simulação Paralela e Distribuída podem ser exploradas com o uso desta ferramenta, uma vez que seu projeto fornece uma estrutura capaz de receber outras implementações de protocolos para sincronização de processos. Sendo assim, estudos relacionados às abordagens de sincronismo podem ser realizados através da ferramenta, bem como a realização de testes, validações e comparações de outros protocolos.

O fato desta ferramenta ser disponibilizada com código aberto contribui para a evolução dos mecanismos envolvidos na paralelização da Simulação de Eventos Discretos. Pôde-se constatar que o desenvolvimento de aplicações de código aberto, que envolvem a Simulação Paralela, tem-se mostrado um importante meio para a evolução das tecnologias relacionadas às ferramentas de Simulação de Eventos Discretos.

Outro tipo de contribuição está relacionado aos conteúdos técnicos e científicos apresentados nesta dissertação. Desta forma, o estudo e sumarização das pesquisas envolvidas na área de Simulação Paralela e Distribuída, que foram apresentados no capítulo 2, fornecem uma visão geral sobre este tema científico. Em adição, a revisão sobre os trabalhos realizados por pesquisadores do GPESC permite entender o contexto desta linha de pesquisa desenvolvida no grupo, que está diretamente relacionada a este trabalho.

Por sua vez, o estudo realizado sobre as ferramentas de Simulação de Eventos Discretos, apresentado no capítulo 3 desta dissertação, permitiu identificar requisitos para o desenvolvimento da ferramenta CronoSim. A partir deste estudo pode-se notar que CronoSim apresenta características existentes em outras ferramentas, como por exemplo, a utilização de protocolos otimistas e ambiente gráfico para

elaboração de simulações. O principal diferencial em relação às outras ferramentas de simulação disponíveis é a possibilidade de configurar ou selecionar protocolos de sincronização, possibilitando, ainda, o uso por pesquisadores para validação e testes de outros protocolos. O fato desta ferramenta ser disponibilizada com o código aberto contribui com a comunidade científica, pois é possível reutilizá-la ou adaptá-la para outras situações de testes. Embora as demais ferramentas ofereçam recursos para paralelização da simulação, elas não possuem flexibilidade para este tipo de adaptação. Portanto, esta peculiaridade da ferramenta CronoSim é um fator que possibilita analisar formas para melhorar o desempenho de simulações distribuídas.

Em complemento, o estudo que foi apresentado sobre as ferramentas pode, inclusive, ser utilizado como um princípio para novas pesquisas relacionadas à paralelização de ferramentas de SED, além de servir como um guia sobre ferramentas de código aberto. Sobretudo, este estudo pode ser utilizado para avaliar as ferramentas paralelas para Simulação de Eventos Discretos, as quais possuem características distintas.

6.2 Sugestões para Trabalhos Futuros

Este trabalho, por se tratar de uma nova ferramenta distribuída para simulação de eventos discretos, proporciona várias sugestões para trabalhos futuros, entre elas:

- Utilizar implementações de outros protocolos para sincronização, a fim de realizar análise sobre desempenho computacional. Além disso, estudos comparativos de desempenho dos protocolos poderão ser realizados com o uso da ferramenta.
- Adicionar algoritmos de migração de processos de modo a evitar situações de desbalanceamento de carga.

- Desenvolver mecanismos que permitam a troca dinâmica dos protocolos de sincronização, tomando como base o método desenvolvido por Faria (2016), que considera a análise entre os protocolos *Time Warp* e *Rollback* Solidário.
- Acrescentar outras funções de distribuição de probabilidade e de validação estatística e planejamento de testes.
- Realizar estudos comparativos do desempenho e das funcionalidades da ferramenta desenvolvida e de outras ferramentas equivalentes. Além disso, análises detalhadas sobre a usabilidade da ferramenta podem ser realizadas com usuários deste tipo aplicação.
- Realizar estudos em áreas voltadas à Pesquisa Operacional, através da Simulação de Eventos Discretos, utilizando a ferramenta CronoSim.
- Desenvolver um mecanismo de depuração, possibilitando o acompanhamento passo a passo da simulação, isto é, a cada ocorrência de um evento.
- Desenvolver recursos para animação gráfica durante a simulação.
- Implementar recursos gráficos para melhorar a representação de modelos na área de trabalho da ferramenta, de modo a reduzir o tamanho do desenho de modelos grandes, utilizando agrupamentos de componentes.
- Adicionar outras técnicas de modelagem à ferramenta como Redes de Petri e Statecharts.

Referências

ALOMAIR, Y.; AHMAD, I.; ALGHAMDI, A. A review of evaluation methods and techniques for simulation packages. *Procedia Computer Science*, Elsevier, v. 62, p. 249–256, 2015.

ALZRAIEE, H.; ZAYED, T.; MOSELHI, O. Methodology for synchronizing discrete event simulation and system dynamics models. In: *Proceedings of the Winter Simulation Conference*. [S.l.]: Winter Simulation Conference, 2012. (WSC '12), p. 54:1–54:11.

AYAĞ, Z.; SAMANLIOGLU, F.; YÜCEKAYA, A. Intelligent approach to simulation software evaluation. 2012.

AZEVEDO, M. E. C. V. de. *Projeto e Implementação dos Protocolos Otimistas Time Warp e Solidary Rollback para Simulação Distribuída*. Dissertação (Mestrado) — Universidade Federal de Itajubá, 2012.

BABULAK, E.; WANG, M. Discrete event simulation. *Aitor Goti (Hg.): Discrete Event Simulations. Rijeka, Kroatien: Sciyo*, p. 1, 2010.

BANKS, J. et al. Discrete-event system simulation. Prentice Hall, 2010.

BARBOSA, J. P. C. *Uma Ferramenta Paralela para Simulação de Eventos Discretos com Monitoramento Dinâmico de Processos*. Dissertação (Mestrado) — Universidade Federal de Itajubá, 2012.

BARNES JR., P. D. et al. Warp speed: Executing time warp on 1,966,080 cores. In: *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. New York, NY, USA: ACM, 2013. (SIGSIM PADS '13), p. 327–336. ISBN 978-1-4503-1920-1. Disponível em: <http://doi.acm.org/10.1145/2486092.2486134>.

BAUER, P. et al. Efficient inter-process synchronization for parallel discrete event simulation on multicores. In: ACM. *Proceedings of the 3rd ACM Conference on SIGSIM-Principles of Advanced Discrete Simulation*. [S.l.], 2015. p. 183–194.

BONAVENTURA, M.; WAINER, G. A.; CASTRO, R. Graphical modeling and simulation of discrete-event systems with cd++builder. *Simulation*, Society for Computer Simulation International, San Diego, CA, USA, v. 89, n. 1, p. 4–27, jan. 2013.

BOZOGLAN, S.; GüNAL, M. M. A multi-modal discrete-event simulation model for military deployment. *Proceedings of the Operational Research Society Simulation Workshop 2010 (SW10)*, Elsevier, v. 17, n. 4, p. 597–611, 2009.

BRAGARD, Q.; VENTRESQUE, A.; MURPHY, L. Synchronisation for dynamic load balancing of decentralised conservative distributed simulation. In: ACM. *Proceedings of the 2nd ACM SIGSIM/PADS conference on Principles of advanced discrete simulation*. [S.l.], 2014. p. 117–126.

BRUSCHI, S. M. *ASDA: um ambiente de simulação distribuída automático*. Tese (Doutorado) — Universidade de São Paulo, 2002.

BRUSCHI, S. M. et al. An automatic distributed simulation environment. In: IEEE. *Simulation Conference, 2004. Proceedings of the 2004 Winter*. [S.l.], 2004. v. 1.

BRYANT, R. E. Simulation of packet communication architecture computer systems. Massachusetts Institute of Technology, 1977.

BUSS, A. Simkit analysis workbench for rapid construction of modeling and simulation components. Monterey, California: Naval Postgraduate School., 2004.

CAROTHERS, C. D.; PERUMALLA, K. S. On deciding between conservative and optimistic approaches on massively parallel platforms. In: *Proceedings of the Winter Simulation Conference*. [S.l.]: Winter Simulation Conference, 2010. (WSC '10), p. 678–687.

CARVALHO, E. A. de. *Uma análise sobre as métricas para escalonamento dinâmico de processos em simulação distribuída usando protocolos otimistas*. Dissertação (Mestrado) — Universidade Federal de Itajubá, 2013.

CHANDY, K. M.; MISRA, J. Distributed simulation: A case study in design and verification of distributed programs. *Software Engineering, IEEE Transactions on*, IEEE, n. 5, p. 440–452, 1979.

CHWIF, L.; MEDINA, A. *Modelagem e Simulação de Eventos Discretos, 4ª Edição: Teoria e Aplicações*. [S.l.]: Elsevier Brasil, 2014.

CHWIF, L.; MEDINA, A. C. Introdução ao software de simulação simul8. *Anais do XXXVIII Simpósio Brasileiro de Pesquisa Operacional, Goiânia, GO*, 2006.

COULOURIS, G. et al. *Sistemas Distribuídos-: Conceitos e Projeto*. [S.l.]: Bookman Editora, 2013.

CRUZ, L. B. da. *Projeto de Um Framework para o Desenvolvimento de Aplicações de Simulação Distribuída*. Dissertação (Mestrado) — Universidade Federal de ItajubÃ¡, 2009.

DAGKAKIS, G.; HEAVEY, C. A review of open source discrete event simulation software for operations research. *Journal of Simulation*, Nature Publishing Group, 2015.

D'ANGELO, G. Parallel and distributed simulation from many cores to the public cloud. In: IEEE. *High Performance Computing and Simulation (HPCS), 2011 International Conference on*. [S.l.], 2011. p. 14–23.

DEITEL, H. M.; DEITEL, H. *Java: How to Program*. [S.l.]: Prentice Hall Press, 2014.

FARIA, R. C. *Método para Análise de Troca de Protocolos Otimistas em um Ambiente de Simulação Distribuída*. Dissertação (Mestrado) — Universidade Federal de Itajubá, 2016.

FSF, F. S. F. *Licenças*. Jan 2015. Disponível em: <http://www.gnu.org/licenses/>.

FUJIMOTO, R. Parallel and distributed simulation. In: *2015 Winter Simulation Conference (WSC)*. [S.l.: s.n.], 2015. p. 45–59.

FUJIMOTO, R. M. Parallel discrete event simulation. *Communications of the ACM*, ACM, v. 33, n. 10, p. 30–53, 1990.

FUJIMOTO, R. M. *Parallel and distributed simulation systems*. [S.l.]: Wiley New York, 2000.

FUJIMOTO, R. M. Research challenges in parallel and distributed simulation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, ACM, v. 26, n. 4, p. 22, 2016.

FUJIMOTO, R. M.; MALIK, A. W.; PARK, A. Parallel and distributed simulation in the cloud. *SCS M&S Magazine*, v. 3, p. 1–10, 2010.

GLINSKY, E.; WAINER, G. New parallel simulation techniques of devs and cell-devs in cd++. In: IEEE COMPUTER SOCIETY. *Proceedings of the 39th annual Symposium on Simulation*. [S.l.], 2006. p. 244–251.

GOLDSMAN, D.; NANCE, R. E.; WILSON, J. R. A brief history of simulation revisited. In: WINTER SIMULATION CONFERENCE. *Proceedings of the Winter Simulation Conference*. [S.l.], 2010. p. 567–574.

GOMES, F. et al. Simkit: a high performance logical process simulation class library in c++. In: IEEE. *Simulation Conference Proceedings, 1995. Winter*. [S.l.], 1995. p. 706–713.

HARRELL, C.; GHOSH, B. K.; BOWDEN, R. O. Simulation using promodel. McGraw-Hill, 2004.

HARVEY, C.; GENTILE, J. E. Synchronization methods for distributed agent based models. *Changing the Face of HPC*, p. 12, 2015.

HOLLOCKS, B. Forty years of discrete-event simulation: a personal reflection. *Journal of the Operational Research Society*, JSTOR, p. 1383–1399, 2006.

HUGHES, E. et al. Introduction to sas simulation studio. In: IEEE PRESS. *Proceedings of the 2013 Winter Simulation Conference: Simulation: Making Decisions in a Complex World*. [S.l.], 2013. p. 4026–4036.

ITEXT. *iText, a JAVA PDF library*. Mar 2016. Disponível em: <https://sourceforge.net/projects/itext/>.

JADHAV, A. S.; SONAR, R. M. Evaluating and selecting software packages: A review. *Information and software technology*, Elsevier, v. 51, n. 3, p. 555–563, 2009.

JAFER, S.; LIU, Q.; WAINER, G. Synchronization methods in parallel and distributed discrete-event simulation. *Simulation Modelling Practice and Theory*, Elsevier, v. 30, p. 54–73, 2013.

JALENDER, B.; GOVARDHAN, A.; PREMCHAND, P. Designing code level reusable software components. *International Journal of Software Engineering & Applications*, Academy & Industry Research Collaboration Center (AIRCC), v. 3, n. 1, p. 219, 2012.

JEFFERSON, D. R. Virtual time. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, ACM, v. 7, n. 3, p. 404–425, 1985.

JGRAPH. *JGraphX (JGraph 6) User Manual*. Abril 2006. Disponível em: <https://jgraph.github.io/mxgraph/docs/manual_javavis.html>.

JOHNSON, R. E.; FOOTE, B. Designing reusable classes. *Journal of object-oriented programming*, v. 1, n. 2, p. 22–35, 1988.

JUNQUEIRA, M. A. F. C. *Mecanismos para Migração de Processos na Simulação Distribuída*. Dissertação (Mestrado) — Universidade Federal de Itajubá, 2012.

KAWABATA, C. L. O. et al. Performance evaluation of a cmb protocol. In: *Proceedings of the 2006 Winter Simulation Conference*. [S.l.: s.n.], 2006. p. 1012–1019.

KELTON, W. D.; SADOWSKI, R. P.; STURROCK, D. T. *Simulation with ARENA. 6/e edn*. [S.l.]: McGraw-Hill, New York, 2014.

KHALILI, M. H.; ZAHEDI, F. Modeling and simulation of a mattress production line using promodel. In: IEEE. *Simulation Conference (WSC), 2013 Winter*. [S.l.], 2013. p. 2598–2609.

KING, D.; HARRISON, H. S. Open-source simulation software jaamsim. In: IEEE PRESS. *Proceedings of the 2013 Winter Simulation Conference: Simulation: Making Decisions in a Complex World*. [S.l.], 2013. p. 2163–2171.

KING, D.; HARRISON, H. S. Jaamsim programming manual. *http://jaamsim.com/docs/JaamSim%20Programming%20Manual%20-%20rev%200.51.pdf*, Apr 2016. Acessado: 16/05/2016.

KRAHL, D. Extendsim 9. In: IEEE PRESS. *Proceedings of the 2013 Winter Simulation Conference: Simulation: Making Decisions in a Complex World*. [S.l.], 2013. p. 4065–4072.

KUNZ, G. et al. Parallel expanded event simulation of tightly coupled systems. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, ACM, v. 26, n. 2, p. 12, 2016.

LANKAMP, M. et al. Mgsim-simulation tools for multi-core processor architectures. *arXiv preprint arXiv:1302.1390*, 2013.

LAPRE, J. M. et al. Time warp state restoration via delta encoding. In: IEEE PRESS. *Proceedings of the 2015 Winter Simulation Conference*. [S.l.], 2015. p. 3025–3036.

MALIK, A. W.; PARK, A. J.; FUJIMOTO, R. M. An optimistic parallel simulation protocol for cloud computing environments. *SCS M&S Magazine*, v. 4, p. 1–9, 2010.

MATTERN, F. Efficient algorithms for distributed snapshots and global virtual time approximation. *Journal of Parallel and Distributed Computing*, Elsevier, v. 18, n. 4, p. 423–434, 1993.

MONTEVECHI, J. et al. Application of design of experiments on the simulation of a process in automotive industry. In: *Winter Simulation Conference*. [S.l.: s.n.], 2007.

MOREIRA, E. et al. Using consistent global checkpoints to synchronize processes in distributed simulation. In: IEEE. *Distributed Simulation and Real-Time Applications, 2005. DS-RT 2005 Proceedings. Ninth IEEE International Symposium on*. [S.l.], 2005. p. 43–50.

MOREIRA, E. M. *Rollback Solidário: um novo protocolo otimista para simulação distribuída*. Tese (Doutorado) — Universidade de São Paulo, 2005.

MOREIRA, E. M. et al. Um framework para o desenvolvimento de programas de simulação distribuída. *Anais do XLII Simpósio Brasileiro de Pesquisa Operacional*, 2010.

MPJ-EXPRESS. *MPJ Express: An Implementation of MPI in Java*. 2014. Disponível em: <http://mpj-express.org/docs/guides/linuxguide.pdf>.

MUNCK, S. D.; VANMECHELEN, K.; BROECKHOVE, J. Revisiting conservative time synchronization protocols in parallel and distributed simulation. *Concurrency and Computation: Practice and Experience*, Wiley Online Library, v. 26, n. 2, p. 468–490, 2014.

NUNES, L. F. et al. Cronosim - uma ferramenta de simulação distribuída de eventos discretos com gerenciamento de processos. In: *Proceedings of the 2015 Brazilian Symposium on Operations Research*. [S.l.: s.n.], 2015. p. 3149–3160.

ONISHI, A. *Técnicas de Reuso de Software aplicados na elaboração de Arquiteturas Corporativas*. [S.l.]: Universidade de São Paulo,[200-], 2006.

PARREIRA JR, W. M. *Apostila de Modelagem e Avaliação de Desempenho: Teoria das Filas e Simulações*. [S.l.: s.n.], 2010. Apostila.

RANGEL, J. J. de A.; CORDEIRO, A. C. A. Free and open-source software for sustainable analysis in logistics systems design. *Journal of Simulation*, Nature Publishing Group, v. 9, n. 1, p. 27–42, 2015.

RAYNAL, M. *Distributed algorithms for message-passing systems*. [S.l.]: Springer, 2013.

RONNGREN, R. et al. A comparative study of state saving mechanisms for time warp synchronized parallel discrete event simulation. In: IEEE. *Simulation Symposium, 1996., Proceedings of the 29th Annual*. [S.l.], 1996. p. 5–14.

SAKURADA, N.; MIYAKE, D. I. Aplicação de simuladores de eventos discretos no processo de modelagem de sistemas de operações de serviços. *Gestão & Produção*, SciELO Brasil, v. 16, n. 1, p. 25–43, 2009.

SARKAR, N. I.; HALIM, S. A. A review of simulation of telecommunication networks: simulators, classification, comparison, methodologies, and recommendations. Cyber Journals, 2011.

SWAIN, J. *INFORMS Simulation software survey*. 2015. Disponível em: <http://www.orms-today.org/surveys/Simulation/Simulation.html>.

SWENSON, B. P.; IVEY, J. S.; RILEY, G. F. Performance of conservative synchronization methods for complex interconnected campus networks in ns-3. In: IEEE PRESS. *Proceedings of the 2014 Winter Simulation Conference*. [S.l.], 2014. p. 3096–3106.

TANG, S.; LEE, B.-S.; HE, B. Speedup for multi-level parallel computing. In: IEEE. *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*. [S.l.], 2012. p. 537–546.

TEWOLDEBERHAN, T. W. et al. Software evaluation and selection: an evaluation and selection methodology for discrete-event simulation software. In: WINTER SIMULATION CONFERENCE. *Proceedings of the 34th conference on Winter simulation: exploring new frontiers*. [S.l.], 2002. p. 67–75.

THIESING, R. M.; PEGDEN, C. D. Introduction to simio. In: IEEE PRESS. *Proceedings of the 2013 Winter Simulation Conference: Simulation: Making Decisions in a Complex World*. [S.l.], 2013. p. 4052–4061.

VARGA, A. The omnet++ discrete event simulation system. version 4.3. user manual. *URL: http://www. omnetpp. org*, 2013.

VAZ, R. da S. *Simulação Distribuída em Cloud Computing*. Dissertação (Mestrado) — Universidade Federal de Itajubá, 2015.

WANG, Y.-M. Consistent global checkpoints that contain a given set of local checkpoints. *Computers, IEEE Transactions on*, IEEE, v. 46, n. 4, p. 456–468, 1997.

WEINGÄRTNER, E.; LEHN, H. V.; WEHRLE, K. A performance comparison of recent network simulators. In: IEEE. *Communications, 2009. ICC'09. IEEE International Conference on*. [S.l.], 2009. p. 1–5.

YILMAZ, L. et al. Panel: The future of research in modeling amp; simulation. In: *Proceedings of the Winter Simulation Conference 2014*. [S.l.: s.n.], 2014. p. 2797–2811. ISSN 0891-7736.

ZEHE, D. et al. Tutorial on a modeling and simulation cloud service. In: IEEE PRESS. *Proceedings of the 2015 Winter Simulation Conference*. [S.l.], 2015. p. 103–114.