Universidade Federal de Itajubá – UNIFEI
Programa de Pós-graduação em Ciência e Tecnologia da Computação

Uma Ferramenta Paralela para Simulação de Eventos Discretos com Monitoramento Dinâmico de Processos

João Paulo Chaves Barbosa

Orientador: Prof. Dr. Edmilson Marmo Moreira

UNIFEI – Itajubá, MG
Junho de 2012

Universidade Federal de Itajubá – UNIFEI
Programa de Pós-graduação em Ciência e Tecnologia da Computação

Uma Ferramenta Paralela para Simulação de Eventos Discretos com Monitoramento Dinâmico de Processos

João Paulo Chaves Barbosa

Orientador: Prof. Dr. Edmilson Marmo Moreira

Dissertação apresentada ao Programa de Pós-graduação em Ciência e Tecnologia da Computação, da Universidade Federal de Itajubá, como parte dos requisitos para a obtenção do título de Mestre em Ciências – Área de Ciência e Tecnologia da Computação.

UNIFEI – Itajubá, MG
Junho de 2012

Ficha catalográfica elaborada pela Biblioteca Mauá –
Bibliotecária Margareth Ribeiro- CRB_6/1700

B234f

Barbosa, João Paulo Chaves
Uma ferramenta paralela para simulação de Eventos Discre_ tos com monitoramento dinâmico de processos / João Paulo Cha_ ves Barbosa. -- Itajubá, (MG) : [s.n.], 2012.
59 p. : il.

Orientador: Prof. Dr. Edmilson Marmo Moreira.
Dissertação (Mestrado) – Universidade Federal de Itajubá.

1. Simulação distribuída. 2. Ambiente Modelagem. 3. Moni_ toramento de processos. 4. Computação paralela. I. Moreira, Ed_ milson Marmo, orient. II. Universidade Federal de Itajubá. III. Título.

Agradecimentos

A Deus que conduz os nossos passos.

Ao Prof. Edmilson Marmo Moreira pela orientação, dedicação, cooperação, incentivo nos momentos difíceis e pela paciência tida comigo.

Aos meus pais, Waquir e Maristela, pelo carinho, amor, incentivo e dedicação.

À minha irmã Rita de Cássia, pelo incentivo, amizade e carinho.

Ao Prof. Marcos Alberto de Carvalho, pela amizade e incentivo.

Ao Prof. Otávio Augusto Salgado Carpinteiro pela cooperação e paciência.

A Prime Interway pelo incentivo, flexibilidade na minha carga horária e caminhos abertos na minha carreira profissional.

Ao Rubens Bertolo Marques pelo incentivo e confiança.

A toda equipe de TI da Prime Interway (*Mírian e Rodrigo*) pelo incentivo e amizade.

Aos colegas que contribuíram, direta ou indiretamente, com este trabalho.

À minha família pelo carinho, alegria e incentivo.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Esta dissertação apresenta o projeto de uma ferramenta para simulação de eventos discretos que tem como objetivo oferecer aos usuários de simulação um ambiente integrado para: *modelagem*, *simulação distribuída* e *monitoramento de processos*. A modelagem conta com um ambiente gráfico que possibilita ao usuário construir um sistema utilizando as linguagens simbólicas (*Redes de Petri, Redes de Fila ou Statecharts*) presentes na ferramenta. A simulação distribuída ocorre de maneira transparente para o usuário, pois a ferramenta provê mecanismos para o tratamento de erros de causa e efeito, podendo o usuário escolher um dentre dois protocolos otimistas para sincronização, são eles: *Time Warp* e *Rollback Solidário*. Para que a simulação ocorra no ambiente distribuído, são utilizadas duas bibliotecas, que permitem a troca de mensagens entre os processos: *PVM (Parallel Virtual Machine)* e *MPI (Massage Passing Interface)*, podendo o usuário escolher uma para a utilização. A fim de se obter um melhor desempenho da simulação distribuída, são utilizados recursos de *escalonamento de processos* e *balanceamento de carga*, que, juntamente com o mecanismo de monitoramento, conseguem equilibrar o uso de carga entre os processadores. Para o desenvolvimento desta ferramenta, foi utilizado um *framework* voltado para o desenvolvimento de programas de simulação distribuída, desenvolvido no *Grupo de Pesquisa em Engenharia de Sistemas e de Computação* (GPESC) da UNIFEI. Para atender as necessidades não inclusas no *framework*, foi necessário estendê-lo, possibilitando, assim, a conclusão deste trabalho.

Abstract

This paper presents the design of a tool for discrete event simulation that offers users an integrated environment for modeling, distributed simulation, and processes monitoring. The tool has a graphic environment that allows the user building a model using the symbolic languages present in the tool (Petri Nets, Queue Nets, and Statecharts). The distributed simulation occurs transparently to the user, since the tool provides mechanisms to handle errors of cause and effect, through the optimistic synchronization protocols: Time Warp and Solidary Rollback. Furthermore, the tool performs the exchange of messages between processes through the libraries: Parallel Virtual Machine (PVM) and Massage Passing Interface (MPI), whose choice is the responsibility of the user. In order to obtain a better performance of distributed simulation, processes scheduling and load balancing are used with the monitoring mechanism to improve the performance by reducing the occurrence of rollbacks.

Capítulo 1

Introdução

A simulação computacional começou a ser utilizada efetivamente no começo da década de 60. A primeira área a utilizar este método como ferramenta de análise e planejamento foi a militar, nos EUA. Com o sucesso obtido na aplicação do método na área militar, sua extensão à indústria norte-americana ocorreu rapidamente, proporcionando o desenvolvimento do método através da evolução das linguagens de programação (CASTILHO, 2004).

Para a criação de uma simulação, o primeiro passo é a obtenção de um modelo, que é definido como sendo uma abstração das relações e comportamentos das componentes de um sistema real, assim provendo os dados necessários para execução do sistema desejado.

A simulação exige a manipulação de grande quantidade de dados para sua execução, sendo estes dados responsáveis pela representação do sistema real. Desta maneira, quanto maior o sistema a ser simulado, maior é o volume de dados requerido para sua simulação.

A necessidade de simulações de sistemas grandes e complexos, que demandam a alocação de muitos recursos computacionais, pode tonar desvantajosa a execução de um programa de simulação em um ambiente sequencial.

A fim de minimizar o tempo de execução da simulação, foram adotadas abordagens paralelas com o objetivo de melhorar o desempenho e assim diminuir o tempo de execução da simulação. No cenário paralelo, os processos da simulação são divididos em vários processos menores, sendo chamados de **processos lógicos**, e cada um desses processos lógicos é enviado ao nó da máquina paralela ou de um sistema distribuído para que, dessa forma, seja executado em paralelo aos demais. Neste contexto, o conceito de Simulação Distribuída vem sendo desenvolvido com o objetivo de diminuir o tempo de execução de um programa de simulação (BRUSCHI, 2003).

1.1 Motivação

A realização de uma simulação, mesmo em um ambiente sequencial, é uma tarefa difícil, pois exige do usuário conhecimentos sobre: modelagem, programação e matemática (probabilidade e estatística), tornando assim, para novos ou inexperientes usuários, árdua a tarefa de criação, validação, executação e análise dos resultados obtidos.

Em um ambiente distribuído, essa atividade pode se tornar ainda mais difícil, já que, além dos conhecimentos citados anteriormente, é necessário que o usuário possua conhecimentos de Computação Paralela e de Sistemas Distribuídos, que não são fáceis nem rápidos de se obter.

Neste sentido, a utilização de um ambiente de simulação, que apresente uma interface visual, suporte a execução distribuída de forma transparente e que permita aos seus usuários maior agilidade no processo de criação de modelos e facilidade para execução da simulação em um ambiente distribuído, é de grande interesse.

Da mesma forma, o desenvolvimento de ferramentas para o monitoramento de processos permite que os recursos alocados nos nós sejam monitorados e, eventualmente, reescalonados, buscando aumentar o desempenho do programa de simulação.

Por conseguinte, as características citadas anteriomente podem contribuir de maneira significativa para o uso da simulação por parte de usuários iniciantes e, principalmente, dos mais experientes que não utilizam a abordagem distribuída.

1.2 Objetivos

Procurando oferecer uma maior agilidade e facilidade para os usuários da simulação distribuída, essa dissertação apresenta o projeto de uma ferramenta visual interativa para auxiliar o usuário desde a fase de modelagem 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.

Com a adoção dos conceitos de computação distribuída, sendo aplicada de forma transparente para o usuário, é possível simular modelos complexos em um menor tempo. A ferramenta de monitoramento que será apresentada possibilitará ao usuário ter conhecimento sobre os recursos utilizados pelos processos distribuídos, podendo, inclusive, intervir no escalonamento.

1.3 Estrutura da Dissertação

Este trabalho está organizado da seguinte forma: o próximo capítulo apresenta um estudo introdutório sobre Simulação Distribuída, sendo discutidas as principais características, benefícios e dificuldades encontradas para desenvolver programas de simulação para este tipo de arquitetura.

Uma revisão da literatura sobre ambientes de modelagem para simulação distribuída é apresentada no capítulo 3, abordando as definições sobre o que é um ambiente de modelagem e suas principais características. É apresentado também um estudo comparativo entre as principais ferramentas disponíveis no mercado para modelagem e simulação, aspectos como: recursos disponíveis, usabilidade e capacidade de simulação, são tratados.

O capítulo 4 aborda os conceitos básicos relacionados ao monitoramento de processos. Com base nesse estudo, é possível implementar recursos para vizualizar os atributos dos nós de uma arquitetura paralela e, também, acompanhar passo a passo a evolução da simulação.

No capítulo 5 é apresentado a modelagem das principais classes da ferramenta proposta. Como pode ser verificado neste capítulo, foi adotada a utilização do padrão arquitetônico em camadas, na qual cada camada exerce funções específicas à sua atribuição.

O capítulo 6 traz uma discussão sobre a implementação do primeiro protótipo da ferramenta. Nele são discutidas as principais características e funcionalidades projetadas para a ferramenta.

Finalmente, são apresentadas as conclusões deste trabalho e as sugestões para a continuação do mesmo.

Capítulo 2

Simulação Distribuída

A simulação é uma ferramenta de grande importância para aquisição de conhecimentos sobre o comportamento de um sistema. É um método numérico de resolução de problemas e consiste na observação ao longo do tempo do desempenho de um modelo que representa um sistema definido a partir do problema a ser resolvido. De forma a representar os sistemas reais em uma abstração computacional, a utilização de modelos permite que sejam construídos diferentes cenários de um sistema, onde cada modelo pode representar uma situação, estado determinado ou uma solução esperada, fornecendo assim um ambiente de experimentação entre os modelos criados. Desse modo, também funciona como uma importante ferramenta para tomada de decisões, como pode ser visto em Castilho (2004), minimizando o risco de erros.

2.1 Simulação Distribuída

Com a necessidade de simular sistemas cada vez maiores e mais complexos, onde o uso da abordagem sequencial pode tornar a simulação inviável, foram desenvolvidos diversos mecanismos que permitem o uso da computação paralela e/ou distribuída. Nesta abordagem, os resultados são alcançados mais rapidamente e, além disso, os modelos complexos podem ser tratados. Ainda neste contexto, pesquisas recentes têm avançado no desenvolvimento de novas técnicas para adaptar os programas de simulação distribuída para arquiteturas computacionais mais recentes, tais como grades (*Grids*) computacionais e computação nas nuvens (*Cloud Computing*) (PARK; FUJIMOTO, 2007; FUJIMOTO; MALIK; PARK, 2010).

O conceito de simulação distribuída consiste na distribuíção dos processos que compõem o programa de simulação entre vários processadores que estejam disponíveis, de forma assim, ocorrer o processamento paralelo da simulação. Segundo Yau (1999), exitem duas abordagem que podem ser utilizadas para a distribuir estes processos, são elas:

MRIP (*Multiple Replication in Parallel*) e SRIP (*Single Replication in Parallel*). Na abordagem SRIP, o programa é dividido em pequenos processos lógicos menores que serão alocados e executados de forma que as operações tenham condições de serem processadas paralelamente. Com a abordagem MRIP, o programa já não é mais particionado, ao invés disto, replicações independentes de um mesmo programa de simulação sequencial são executadas em paralelo. Cada replicação envia seus resultados (variáveis estimadas) para um analisador global, onde as médias finais são calculadas.

A implementação de uma simulação distribuída 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.

Um erro de causa e efeito acontece quando um evento é executado fora de seu tempo lógico, esse comportamento pode afetar o restante da simulação, pois, segundo Chandy e Misra (1979), para garantir a correctude dos resultados, em um programa de simulação de eventos discretos distribuído, é suficiente executar os eventos em ordem cronológica. Desta forma, uma maneira de garantir a ordem de execução cronológica dos eventos é utilizando protocolos que garantam o sincronismo entre os processos da simulação. Os procotolos de sincronismo que foram desenvolvidos são divididos em duas classes: conservativos e otimistas, evitando ou corrigindo erros de causa e efeito, respectivamente (FUJIMOTO, 2003).

2.1.1 Protocolos Conservativos

Os primeiros protocolos desenvolvidos foram os conservativos, possuindo uma abordagem simples para garantir que um processo lógico executará seus eventos em ordem cronológica. Basicamente, um evento é executado somente quando possuir o menor rótulo de tempo e não houver possibilidade do processo lógico receber algum evento com rótulo de tempo menor que ele.

O procotolo conservativo mais famoso é o CMB, nome dado em homenagem aos seus criadores: Chandy e Misra (1979) e Bryant (1977).

Em um protocolo conservativo, como o CMB, os canais de comunicação são preestabelecidos antes da execução dos processos. Desta maneira, os processos só podem se

comunicar com outros processos da simulação que estiverem conectados a ele. Essa medida ajuda aos processos conhecer o tempo lógico de seus vizinhos e, assim, tratar os eventos que possuam o tempo de ocorrência menor que o tempo lógico dos canais de comunicação.

O processo lógico seleciona repetidamente o canal com o menor LVT (*Local virtual Time*) e, se houver mensagem na fila, executa-a. A ordem de processamento dos eventos estará correta porque todas as mensagens que serão recebidas no futuro terão marcas de tempo com valores maiores que o LVT, uma vez que as mensagens trafegam pelos canais em fila FIFO (*First In First Out*) e, em função disso, irão chegar em ordem cronológica de ocorrência (MOREIRA, 2005).

2.1.2 Protocolos Otimistas

Para tratar os erros de causa e efeito, os protocolos conservativos impõem uma sincronização explícita entre os processos lógicos (PARK; FUJIMOTO; PERUMALLA, 2004). Nos protocolos otimistas há maior liberdade entre os processos na execução de suas tarefas. O sincronismo ocorre quando há identificação de um erro de causa e efeito. Nessa situação, os processos corrigem o problema através da restauração de um estado anteriormente armazenado, procedimento conhecido como *rollback*. Jefferson (1985) desenvolveu um mecanismo de sincronização denominado *Virtual Time* e propôs um protocolo que se tornou o protocolo otimista mais conhecido, chamado *Time Warp*.

O *Time Warp* é constituído por duas partes distintas: controle local, que é interno a cada processo e se responsabiliza em garantir que os eventos sejam executados em ordem cronológica, utilizando a política de escalonamento Menor *Timestamp* Primeiro (*Smallest-Timestamp-First policy*); e o controle global, destinado ao gerenciamento de memória e cálculo do *Global Virtual Time* (GVT) (MOREIRA, 2005).

O comportamento de um processo no *Time Warp* é semelhante a de um programa de simulação sequencial. Ele possui uma estrutura de dados denominada lista de eventos, que inclui todos os eventos que foram escalonados mas que ainda não foram processados. Como já explicado anteriormente, a lista de eventos é organizada em ordem cronológica do seu tempo lógico e as mensagens que geraram estes eventos são denominadas mensagens positivas. Os eventos que já foram processados são mantidos em uma lista denominada lista de saída de mensagens, como visto na figura 2.1. Essas mensagens contém uma marca de tempo menor ou igual ao LVT do processo e são denominadas mensagens negativas ou anti-mensagens. Essa lista só é utilizada se for necessária a realização de *rollback*.

Se determinado processo, que está realizando um *rollback* (figura 2.2), enviou mensagens para outros processos com tempo lógico maior que o tempo local do processo para onde ele está retornando, estas mensagens precisam ser canceladas. Assim, o processo

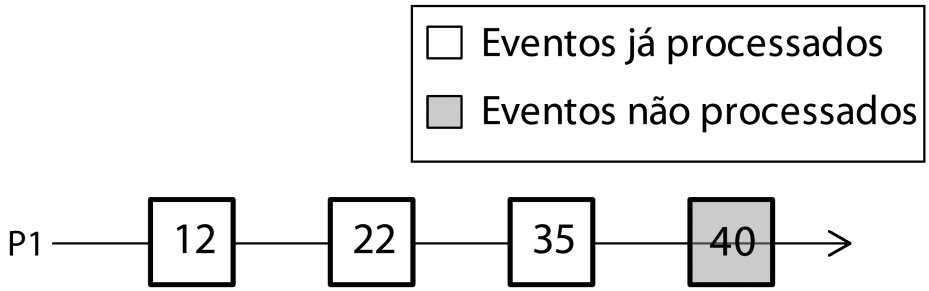

Figura 2.1: Lista de eventos.

envia anti-mensagens avisando do cancelamento dos respectivos eventos. Ao receber uma anti-mensagem, primeiramente o processo verifica se a mensagem correspondente ainda não foi processada e, neste caso, apaga a mensagem da lista de eventos futuros. Se a mensagem já foi processada, o processo também deverá realizar um *rollback*. Esse procedimento reconstitui a simulação a um ponto seguro, entretanto, essa abordagem pode provocar um efeito cascata dos *rollbacks* quando há o envolvimento de diversos processos.

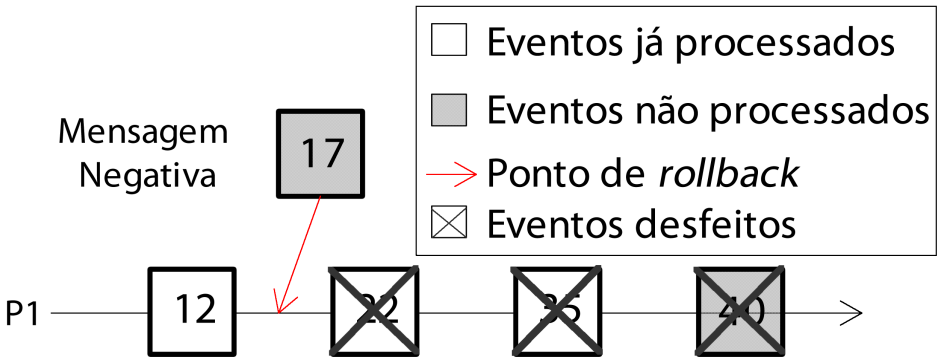

Figura 2.2: Ocorrência de um *rollback*.

O protocolo *Time Warp* evoluiu bastante desde sua criação, devido ao desenvolvimento de diversas pesquisas. Estas pesquisas procuraram melhorar o gerenciamento de memória e minimizar a quantidade de *rollbacks* durante a simulação, através da diminuição do otimismo dos processos da simulação, como, por exemplo, em suas variações: *Breathing Time Warp BTW* e *Local Time Warp LTW* (SPOLON, 2001).

Apesar do *Time Warp* ser o protocolo otimista mais conhecido, Moreira (2005) apresentou um novo protocolo otimista, denominado *Rollback* Solidário e fundamentado na teoria dos *Checkpoints* Globais Consistentes (MOREIRA; SANTANA; SANTANA, 2005), que utiliza uma abordagem diferente para o sincronismo dos processos durante o procedimento *rollback*. No protocolo *Rollback* Solidário, quando uma mensagem com *timestamp* menor que o LVT de algum processo é recebida, o sistema faz um levantamento para identificar quais processos devem realizar um retrocesso e, assim, todos eles, simultaneamente, realizam *rollback*. Neste momento, há a identificação do *checkpoint* global consistente em que os processos deverão retomar a computação. Com a realização de *rollbacks* em

conjunto, para um *checkpoint* global consistente, não existe a necessidade do sistema utilizar anti-mensagens. Desta forma, os processos não necessitam armazenar cópias das mensagens e o consumo de memória é menor neste protocolo, apesar de sua implementação ser mais complexa (MOREIRA, 2005). Além disso, como a decisão do retorno envolve todos os processos, não há o efeito cascata de *rollbacks*, como ocorre no protocolo *Time Warp*.

Independente do protocolo otimista, é importante destacar que nesta abordagem os processos armazenam cópias de suas estruturas internas, para viabilizar a execução dos *rollbacks*. Portanto, o gerenciamento de memória nos programas que implementam estes protocolos é muito importante, pois, dependendo da complexidade do modelo a ser simulado, vários estados serão criados durante a execução do programa, consumindo rapidamente este recurso (memória). Desta forma, é necessário um mecanismo que gerencie a memória e possa recuperar o espaço ocupado por eventos que não serão mais utilizados. Essa situação é conhecida como coleta de fóssil. Assim, os protocolos otimistas implementam um mecanismo que calcula um valor que representa um limite inferior sobre o tempo da simulação, que não está sujeito a ser refeito devido a uma situação de *rollback*. Este mecanismo é conhecido como *Global Virtual Time* (GVT). Portanto, um evento não poderá ser desfeito durante a ocorrência de um *rollback*, tem-se a garantia de que um evento com *timestamp* menor que o GVT não precisará ser desfeito.

A tabela 2.1, apresenta as principais características entre os protocolos: *CMB, Time Warp e Rollback solidário*

Característica	Time Warp	RollBack Solidário	CMB
Otimista	Sim	Sim	Não
Conservativo	Não	Não	Sim
Armazenamento de anti-mensagens	Sim	Não	Não se aplica.
Envio de anti-mensagens	Sim	Não	Não se aplica.
Checkpoints globais	não	Sim	Não se aplica.
Rollbacks	Não	Sim	Não se aplica.
Rollbacks em cascata	Sim	Não	Não se aplica.
Alto trafego de mensagens	Sim	Não	Não se aplica.

Tabela 2.1: Principais características entre os protocolos citados anteriormente

2.2 Considerações finais

A simulação é uma importante ferramenta para análise do comportamento de vários tipos de sistemas. O emprego da simulação exige a manipulação de dados que podem variar em tamanho de acordo com a complexidade dos sistemas reais, sendo que, quanto maior e mais complexo é o sistema a ser simulado, maior será a quantidade de recur-

sos computacionais necessários para sua execução, aumentando assim o seu tempo de processamento.

Por sua vez, a utilização da computação distribuída tem como objetivo diminuir o tempo de execução de grandes processos, atráves da sua execução paralela. Isso significa que um grande programa será dividido em n processos menores que, por sua vez, serão executados paralelamente em diferentes processadores que compõem um sistema distribuído ou uma máquina paralela, melhorando o desempenho do programa.

Por conseguinte, a união da simulação com a computação distribuída criou uma nova área de pesquisa, cujo principal foco é a melhoria do desempenho de um programa de simulação. Os problemas inerentes à computação paralela tiveram, ao longo dos últimos anos, um tratamento especializado, quando tratados em relação aos programas de simulação. Devido às suas características e necessidade de conhecimentos especiais, vários usuários de simulação não utilizam o potencial da simulação distribuída. Neste sentido, o desenvolvimento de ferramentas que forneçam estes recursos é bastante motivador.

Capítulo 3

Modelagem e Ambientes Para Simulação Distribuída de Eventos Discretos

A criação de um modelo é o primerio passo de uma simulação, seja em uma abordagem sequencial ou paralela. O modelo pode ser tratado como uma abstração capaz de representar, em um sistema computacional, o comportamento preciso de um sistema real, para isso é utilizada uma linguagem formal, que possibilite expressar tais abstrações. Algumas das linguagens que podem ser utilizadas para essa representação são: Redes de Petri (MURATA, 1989), *Statecharts* (HAREL, 1987), Teoria de Filas (COSTA, 2006), entre outras, que, através de seus componentes, podem expressar as atividades e fluxos desejadas na modelagem e, consequentemente, na simulação. O grau de abstração do modelo é estabelecido de acordo com os objetivos desejados da simulação, contendo, assim, as características necessárias para representar esses objetos.

Segundo Chwif e Medina (2007), exsitem três categorias básicas de modelos:

Modelos Simbólicos ou Diagramáticos: um modelo simbólico (ou diagramático ou icônico) é composto por símbolos gráficos que representam um sistema de maneira estática, como uma "foto" (sem considerar seu comportamento no tempo). Um fluxograma de processo pode ser considerado como um modelo simbólico. As grandes limitações desses modelos, além de sua representação estática do sistema, são a falta de elementos quantitativos (medidas de desempenho do sistema, por exemplo) e a dificuldade de se representar muitos detalhes de um mesmo sistema. O modelo simbólico é utilizado principalmente na documentação de projetos e como ferramenta de comunicação.

Modelos Matemáticos ou Analíticos: os modelos matemáticos ou analíticos podem

ser vistos como um conjunto de fórmulas matemáticas, por exemplo, os modelos de Programação Linear ou os modelos analíticos da Teoria de Filas. Na sua grande maioria, estes modelos são de natureza estática (o que não é o caso da Teoria das Filas). Muitos destes modelos não possuem soluções analíticas para sistemas complexos, devendo-se utilizar hipóteses simplificadoras, o que pode comprometer a fidelidade do modelo ao problema real. Por outro lado, devido à natureza destes modelos, a solução é rápida e exata, quando existe solução analítica.

Modelos de Simulação: os sistemas reais, geralmente, apresentam uma maior complexidade, devido, principalmente, à sua natureza dinâmica (que muda seu estado ao longo do tempo) e a sua natureza aleatória (que é regida por variáveis aleatórias). O modelo de simulação consegue capturar com mais fidelidade estas características aleatórias procurando repetir em um computador o mesmo comportamento que o sistema apresentaria quando submetido às mesmas condições de contorno.

O desenvolvimento de *softwares* para simulação de eventos discretos tem evoluído progressivamente desde 1960 (BABULAK; WANG, 2010). Essa evolução sempre procurou facilitar o desenvolvimento da simulação, seja nos primórdios, com a utilização de linguagens de simulação, seja também com a utilização da simulação interativa visual, utilizada desde a década de 80. A abordagem visual permite um desenvolvimento mais amigável, menos confuso e mais ágil, facilidades essas que contribuiram de forma importante para a popularização dessas ferramentas nos meios acadêmicos e empresariais.

3.1 Ambientes para modelagem e simulação

Os ambientes de modelagem são ferramentas geralmente bem munidas de usabilidade gráfica, oferecendo suporte aos usuários para a criação de modelos, bem como à simulação destes modelos em arquiteturas sequenciais e/ou distribuídas.

Uma característica importante de um ambiente é a programação visual, sendo possível a criação do modelo de forma visual e, também, a visualização dos resultados por meio de gráficos ou animações, tornando a tarefa do usuário muito mais prática e vantajosa. Estas características compõem uma simulação visual interativa (BELL; OKEEFE, 1987).

Após a elaboração do modelo, o ambiente tem a tarefa de executar a simulação em questão, podendo essa ser executada de modo sequencial ou paralelo. Caso seja escolhida uma simulação em paralelo, o ambiente deve ser capaz de particionar o modelo em processos e também escaloná-los nos nós de processamento.

Atualmente existem várias ferramentas voltadas para modelagem nas mais variadas áreas, tais como:

ARENA : É um ambiente gráfico integrado de simulação (figura 3.1), baseado na linguagem SIMAM V (*SIMulation ANalysis*). SIMAN é uma linguagem de simulação que modela sistemas discretos, contínuos ou híbridos (BRUSCHI, 2003). O Arena emprega um projeto orientado a objetos para o desenvolvimento do modelo interamente gráfico (MARKOVITCH; PROFOZICH, 1996). O usuário pode criar um modelo posicionando os objetos gráficos, chamados módulos, de forma a representar o sistema desejado. Após a criação do modelo, o Arena gera automaticamente o modelo em código SIMAN, que será utilizado para a execução da simulação. Caso seja necessário algum módulo mais específico para a criação de um modelo, o usuário pode criar esse objeto e usá-lo em sua modelagem.

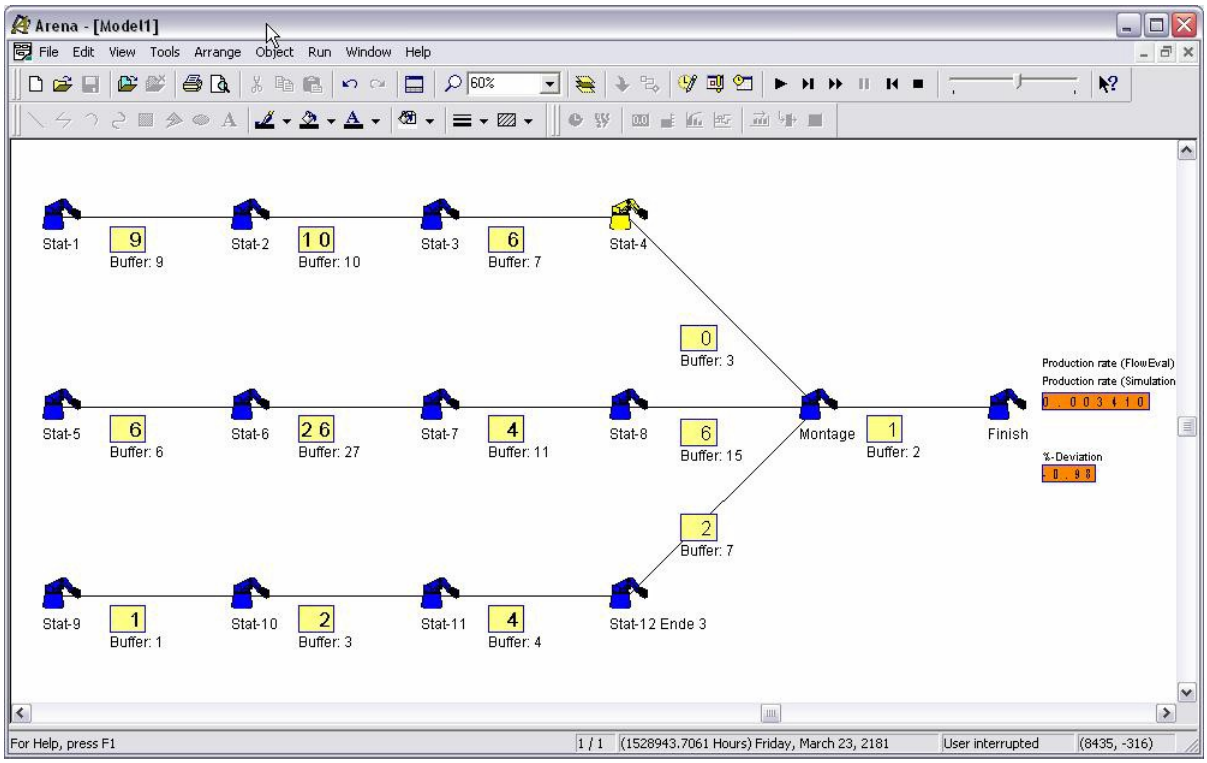

Figura 3.1: Representação de um modelo utilizando o *Software* Arena, retirado de *Advanced-Planning (2012)*

A utilização de *templates* de simulação possibilita a fácil adaptação do Arena para várias áreas diferentes. O Arena possui uma coleção com mais de sessenta módulos, fornecendo recursos de modelagem para vários tipos de aplicações (MARKOVITCH; PROFOZICH, 1996).

Animações no Arena podem ser realizadas simultanemente com a execução do modelo ou em modo pós processamento. Animações podem ser criadas de várias maneiras: através da ferramenta gráfica de desenhos do Arena ou também podem ser criadas pelo AutoCad ou outro programa que gere arquivos no formato DXF, podendo ser importados para o Arena atráves de *Active X* (MARKOVITCH; PROFO-

ZICH, 1996).

Além disso, o recurso oferecido pelo Arena denominado de *Jump-Start Wizard* auxilia o usuário na tarefa de criação do modelo, depois de responder a uma série de perguntas sobre o novo modelo. O assistente *Jump-Start* rapidamente cria um modelo de simulação de nível básico.

AutoMod : Produzido pela *Brooks Software* e oferecido pelo Simul8, o AutoMod (figura 3.2), permite que os usuários construam modelos de qualquer tamanho e complexidade, que podem ser utilizados não só para planejamento e projeto como também para análises das operações diárias e controle de desenvolvimento e testes (ROHRER; MCGREGOR, 2002).

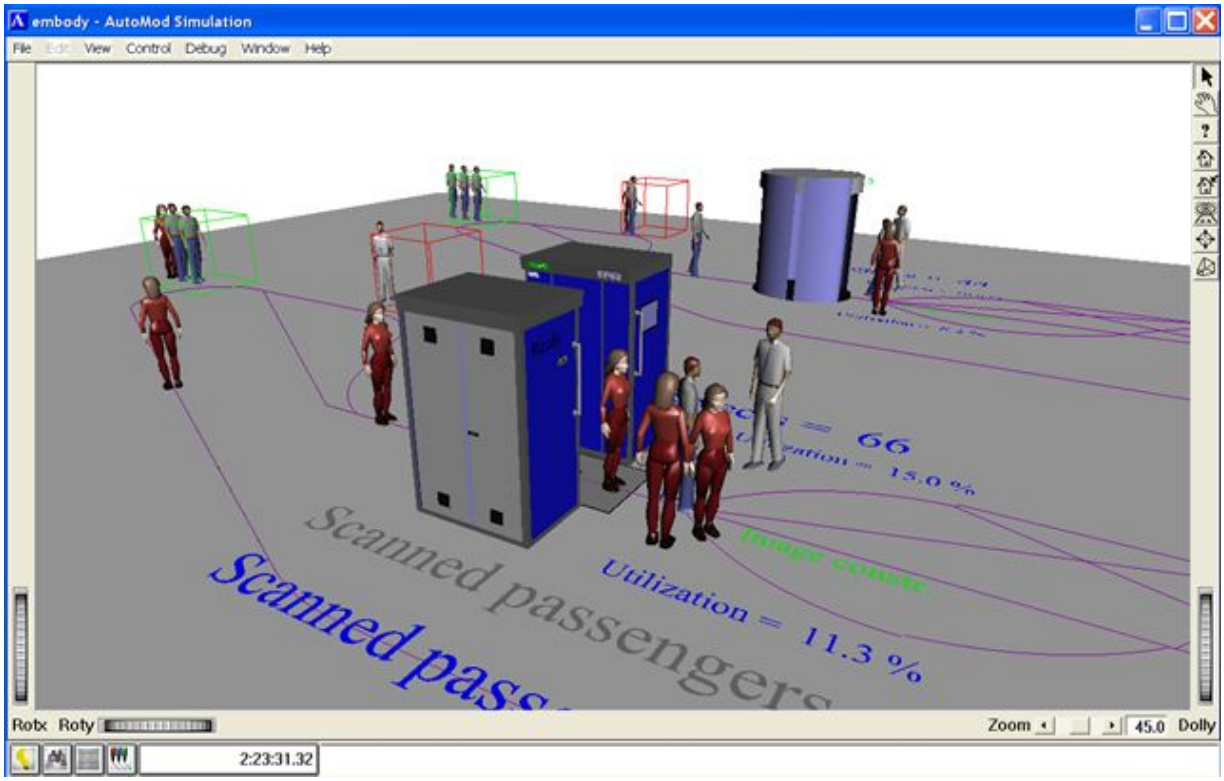

Figura 3.2: Modelo criado utilizando a ferramenta AutoMod Limited (2012)

O AutoMod provê um conjunto de *templates* e objetos que auxiliam a criação de um modelo, tanto em 2D quanto em 3D. A modularidade utilizada na criação de objetos que compõem o modelo, permite a reutilização desses objetos na criação de novos modelos, reduzindo assim o tempo de desenvolvimento.

Na criação de um modelo é possível utilizar os conceitos de funções e subrotinas que podem ser reaproveitadas ou utilizadas em outros modelos. Além disso, o autoMod fornece uma linguagem de programção própria baseada em ações, permitindo o desenvolvimento mais acurado dos processos de um modelo.

ProModel : O *software* de simulação ProModel (figura 3.3), desenvolvido pela PRO-MODEL Corp., possui características voltadas para a manufatura, mas sua flexibilidade de programação permite aplicações em diversas áreas. Law e Kelton (1991) classificam o *software* ProModel como um dos simuladores mais flexíveis devido a sua capacidade de construir lógicas complexas. Apresenta também bons recursos de análise estatística, interface simples e múltiplas replicações de simulação (SAKURADA; MIYAKE, 2003).

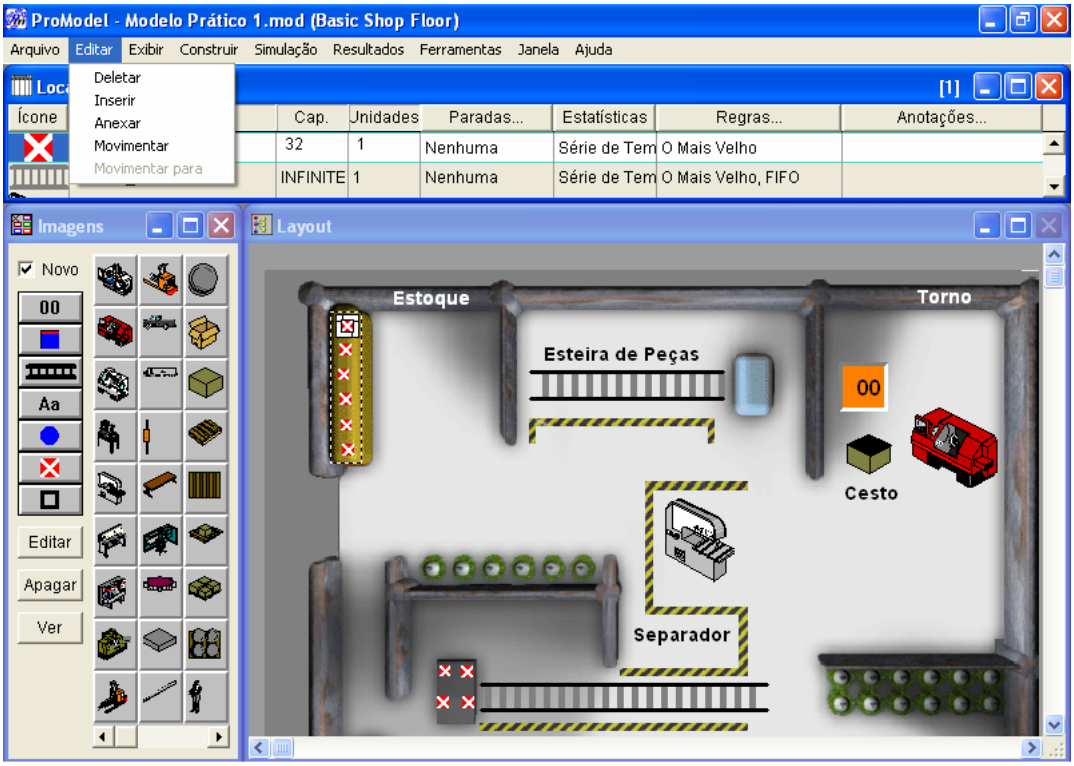

Figura 3.3: Interface da ferramenta Promodel, retirado de Belge (2010)

A geração de aleatoriedade utilizando mais de 20 tipos de distribuições estatísticas ou importação dos dados do usuário, é uma característica importante para a ferramenta, assim como também o seu módulo de depuração e a possibilidade de customização das interfaces.

Witness : O Witness (figura 3.4), oferece um conjunto de ferramentas profissionais para modelar e simular qualquer processo de negócio (WITNESS, 2012).

Pam e Michael (1997) explicam que o Witness é capaz de modelar uma variedade de elementos discretos (*buffers*, peças, máquinas e transportadores) ou contínuos (fluídos, gases e tanques).

A grande variedade de elementos, juntamente com um conjunto abrangente de regras lógicas de controle já existentes no *software*, permitem ao usuário criar um modelo de forma rápida e flexível. Outra característica, importante para o desempenho

na criação de modelos, deve-se a possibilidade do reuso de modelos que podem ser aproveitados utilizando o conceito de herança.

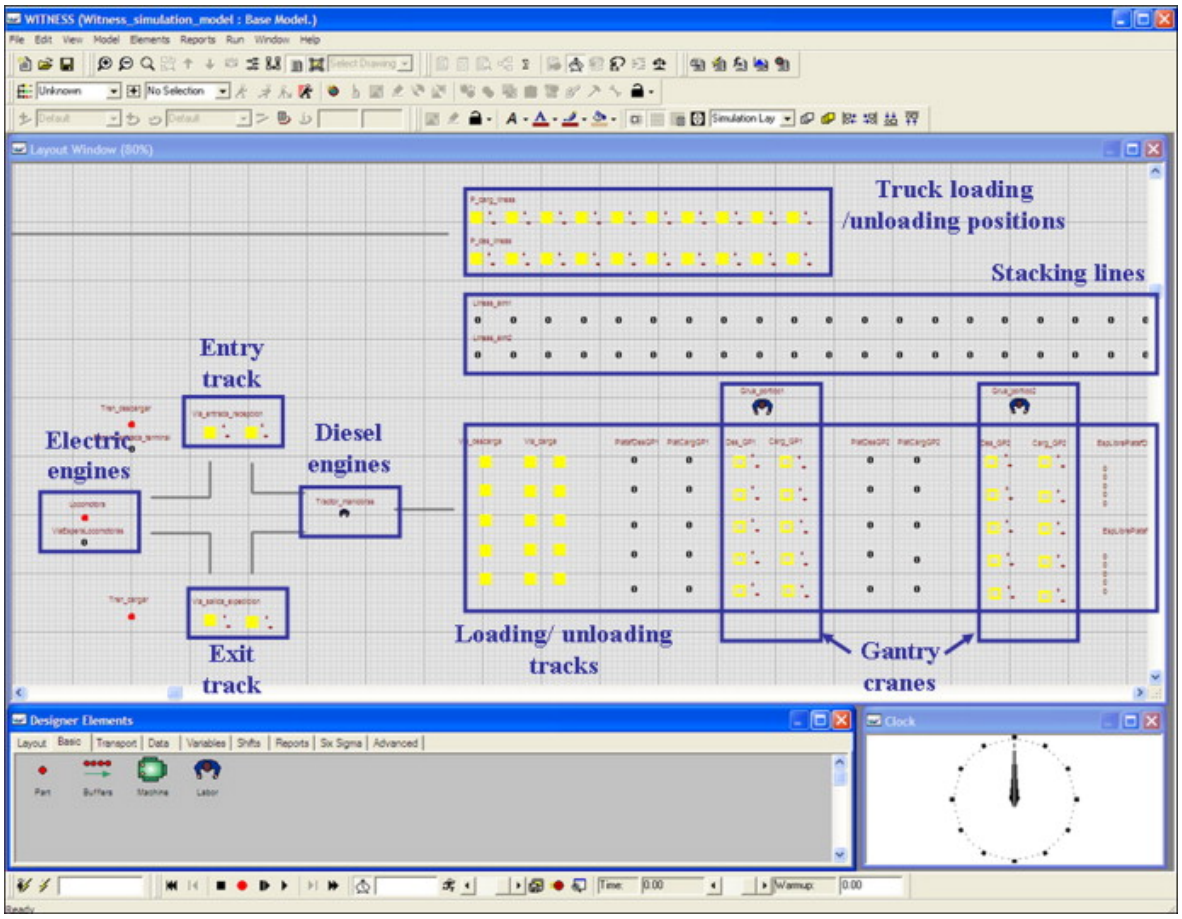

Figura 3.4: Interface da ferramenta Witness, retirado de García e García (2012)

Complementando a simulação, a pós-simulação do Witness oferece relatórios com estatísticas da contagem de elementos, utilização, limites, entre outras análises, auxiliando o usuário na análise dos resultados alcançados durante a simulação.

Em Witness (2012), com a adição de módulos específicos, é possível complementar as áreas de atuação do Witness. Atualmente existem quatro módulos, são eles:

- Automotivo, oferecendo suporte para simulação dentro dos processos e operações convencionadas pela sociedade alemã da indústria automobilística;
- Cadeias de suplementos, auxiliando a modelagem e análise das redes de suplemento, desde cobertura na gestão de cadeias de suplementos e exposição dos potenciais de otimização da logística;
- Estoque, adicionando grande eficiência para criação de simulação de modelos nas divisões de logística;
- Solar, permitindo a modelagem de processos da fabricação de células solares que utilizam a tecnologia de *wafer* e de camadas finas.

SIMIO : SIMIO (*SIMulation Intelligent Objects*) (figura 3.5), é um *framework* de modelagem e simulação baseado em objetos inteligentes. Os objetos inteligentes criados pelo usuário podem ser reutilizados em vários projetos de modelagem. Os objetos podem ser armazenados em bibliotecas e facilmente compartilhados. Um usuário novato pode utilizar objetos pré-construídos existentes em uma biblioteca (PEGDEN, 2007).

Como os outros *softwares* da área, o SIMIO faz modelagem de forma simples, fornecendo uma nova abordagem baseada em objetos (SIMIO, 2012). Permite a seleção de objetos armazenados em bibliotecas, colocando-os graficamente no modelo do usuário. Os objetos representam os componentes físicos do modelo, como estações de trabalho, transportadores, empilhadeiras, macas etc. A criação de animações em 3D, assim como o tratamento para eventos aleatórios, estão presentes na ferramenta. O objeto (componente) no SIMIO adota os conceitos de orientação a objetos, permitindo aos usuários criarem novos objetos sem a necessidade de codificação.

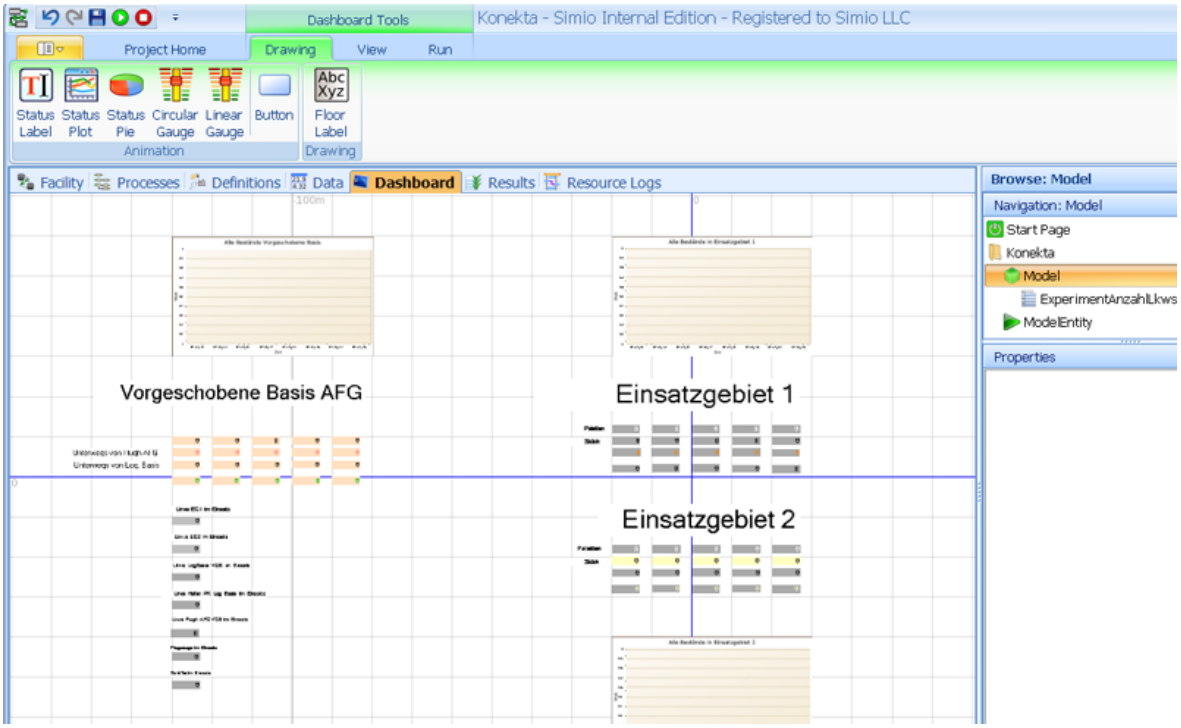

Figura 3.5: Interface da ferramenta SIMIO, retirado de SIMIO (2012)

Algumas características presentes no *software* permitem a sua utilização por profissionais experientes ou, até mesmo, usuários novatos com pouca experiência (SIMIO, 2012).

Simul8 : É um *software* que permite a modelagem e simulação de eventos discretos. O Simul8 (figura 3.6), não diferenciando dos demais *softwares* do mercado, utiliza uma interface de fácil entendimento e que permite aos usuários melhor agilidade no processo de criação do modelo e alteração das propriedades de um objeto.

A utilização de um assistente passo a passo auxilia os novos usuários a criarem lógicas de fluxo pertecentes a um modelo, utilizando a linguagem de *script* estrututada denominada "*event-drive*" e dando ao modelo em questão uma maior precisão.

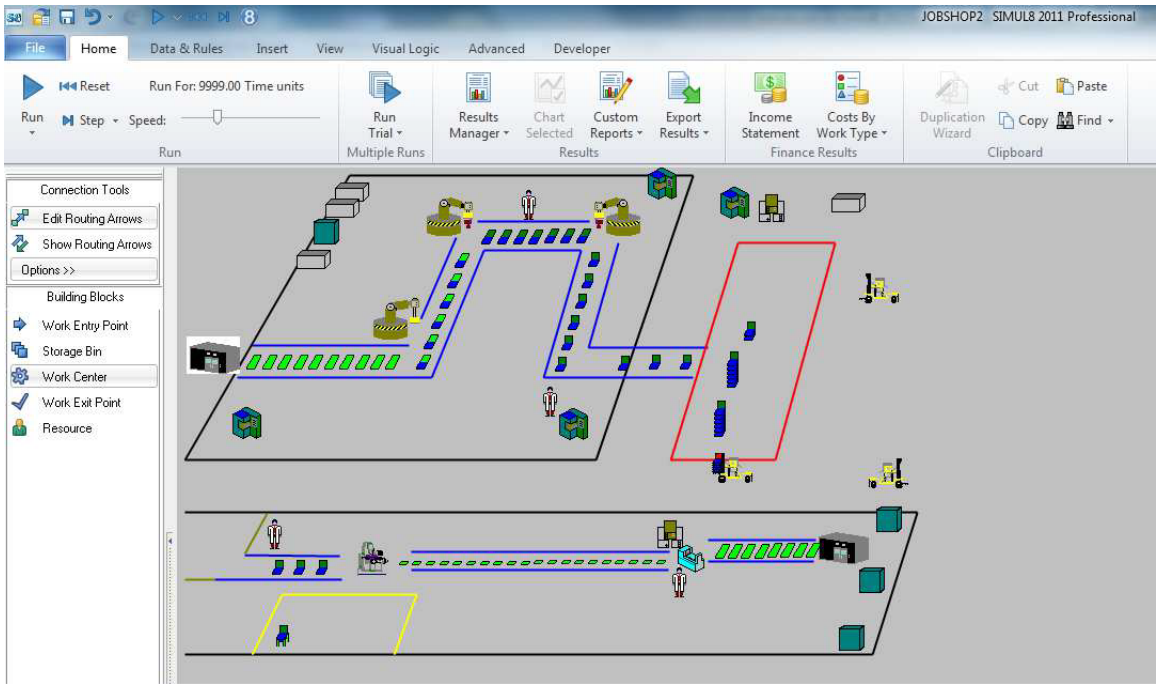

Figura 3.6: Interface da ferramenta Simul8, retirado de Chwif e Medina (2006)

Os recursos gráficos oferecidos com o *software* possibilitam o acompanhamento e a avaliação do sistema através da visualização da formação de filas.

O Simul8 incorpora uma tecnologia de processamento paralelo, possibilitando que uma mesma simulação seja executada em mais de um microcomputador (CHWIF; MEDINA, 2006).

Enterprise Dynamics : O *Enterprise Dynamics* (figura 3.7) é um *software* para modelagem, simulação e controle de processos dinâmicos orientados a objetos. Os usuários podem selecionar objetos, chamados **átomos**, da biblioteca padrão, a fim de construir seu próprio modelo.

O *Enterprise Dynamics* é baseado no conceito de átomos como objetos modelados em cada modelo (DYNAMICS, 2006). Átomos imitam certos objetos da vida real e podem ser organizados para simular um processo da vida real ou sistema específico. Os átomos são agrupados em grupos específicos, chamados *suites*.

Uma plataforma de simulação orientada a objetos tem duas grandes vantagens: objetos independentes podem ser facilmente reutilizados em outros modelos de simulação e o comportamento é definido no objeto em si e não em uma parte diferente do modelo (o que torna o modelo mais claro de entender). Em uma linguagem orientada a eventos, na maioria dos casos, todos os comportamentos possíveis são

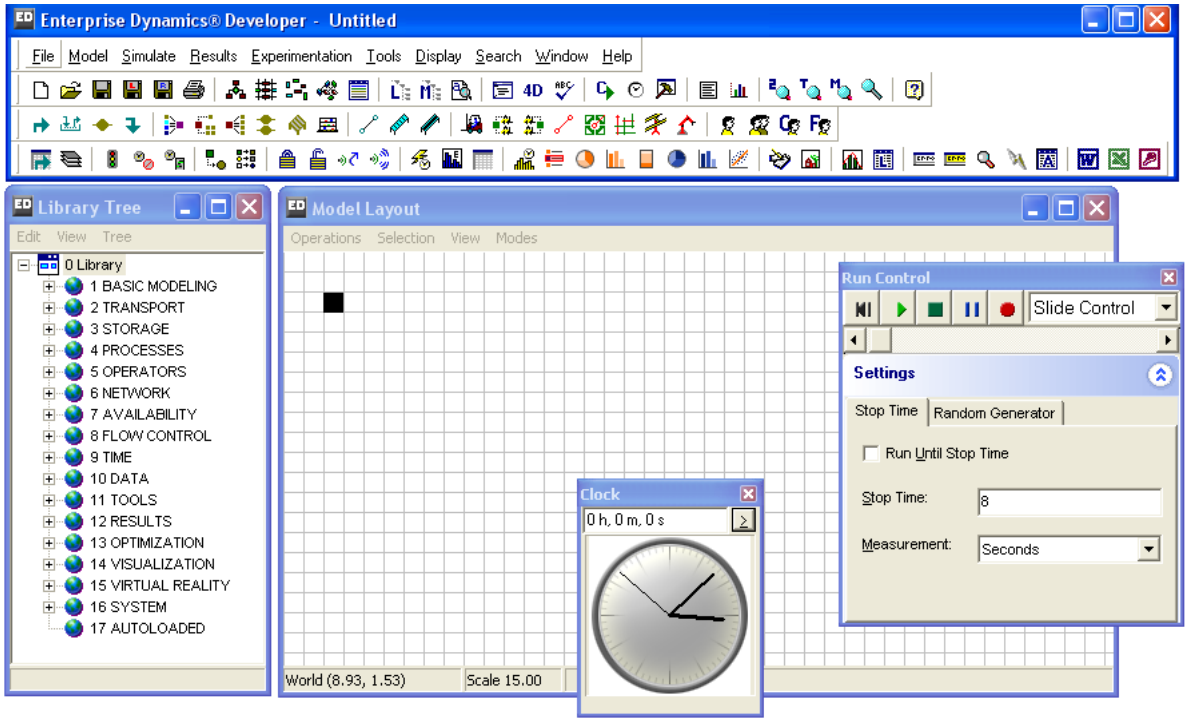

Figura 3.7: Interface da ferramenta *Enterprise Dynamics*, retirado de INCONTROL (2009)

programados em qualquer lugar do programa , isso além de criar um código confuso, também diminui o desempenho.

ASDA : É uma proposta de um ambiente que possibilita a diversos usuários, com os mais diferentes níveis de conhecimento sobre simulação, gerarem modelos para serem simulados de forma distribuída, tando na abordagem MRIP (*Multiple Replication in Parallel*) quanto na SRIP (*Single Replication in Parallel*), de forma mais eficiente. Essa característica é alcançada graças a sua interface amigável, que simplifica a criação e análise dos dados obtidos. Sua principal aplicação é na área de sistemas computacionais (BRUSCHI, 2003).

Essa ferramenta tem como objetivo ser uma alternativa aos caros produtos comerciais da área de simulação. Foi desenvolvido para simular principalmente redes de computadores, multiprocessadores e outros sistemas distribuídos, podendo ainda ser utilizado para simular outros tipos de sistemas (BRUSCHI, 2003).

A tabela 3.1 exibe um resumo das principais características dos *softwares* citados anteriormente.

Característica	Arena	AutoMod	ProModel	Witness	SIMIO	Simul8	E.D	ASDA
Simulação de eventos discretos	X	X	X	X	X	X	X	X
Ambiente gráfico	X	X	X	X	X	X	X	X
Animações da simulação	X	X	X	X	X	X	X	-
Suporte a criação de modelos	X	X	X	X	X	X	X	-
Simulação paralela	-	-	-	-	-	X	-	X
Simulação sequencial	x	X	X	X	X	X	X	-
Monitoramento dos processos	-	-	-	-	-	-	-	-
Escalonamento dos processos	-	-	-	-	-	-	-	-
Templates de modelos	X	X	X	X	X	X	X	-
Estatistícas sobre a simulação	X	X	X	X	X	X	X	X
Biblioteca de objetos	X	X	X	X	X	X	X	-

Tabela 3.1: Comparativo entre os *software* citados e suas principais características

3.2 Linguagens para Modelagem e Simulação

Inicialmente, os sistemas de simulação foram desenvolvidos sobre linguagens de programação de propósito geral (Basic, Pascal, C etc.). No entanto, isto demandava um grande esforço para construção de modelos, além de exigir do profissional responsável pela simulação conhecimentos profundos de programação de computadores. Foi então que apareceram as linguagens de programação de computadores dedicadas à simulação, tais como GPSS, SIMAN, SLAM, SIMSCRIPT etc. Estas linguagens na realidade eram bibliotecas compostas de conjuntos de macro-comandos de outras linguagens de propósito geral (LOBãO; PORTO, 1999).

A seguir uma breve descrição das mais conhecidas:

GPSS : A linguagem *General Purpose Simulation System* foi desenvolvida por Geoffrey Gordon para ser executada em computadores da IBM. Baseava-se em um diagrama de blocos similar a um fluxograma. Em 1965, rodando sobre um IBM 2250 dotado de um terminal interativo, permitia a interrupção do processo para a exibição de resultados intermediários, o que foi um fato marcante para a simulação. No entanto, os altos custos envolvidos impediram seu uso de forma mais abrangente (SCHNEIDER, 2004). O código para simulação a seguir ilustra a simulação de um modelo de fila M/M/1.

```
Simulation of M/M/1 system
SIMULATE
 GENERATE   RVEXPO(1, 2.0)
 QUEUE      SERVQ
 SEIZE      SERVER
 ADVANCE    RVEXPO(2, Q.0)
 RELEASE    SERVER
```

```
 TERMINATE  1
 CONTROL    STATEMENTS
 START  1000
END
```

SIMAN : *SIMulation ANalysis* é um programa para modelagem de sistemas discretos ou combinado discreto/contínuo (DAVIS; PEGDEN, 1988). Segundo Miyagi (2006), SIMAN é uma linguagem de alto nível que, a partir de suas estruturas, permite a construção de modelos, possuindo todos os recursos e facilidades da GPSS, porém com a capacidade de realizar simulação contínua. O código a seguir descreve a simulação de um sistema de filas utilizando esta linguagem.

```
BEGIN;
  CREATE,,EX(2,1);
  QUEUE, 1;
  SEIZE: SERVER;
  DELAY: EX(1,1);
  RELEASE:SERVER:DISPOSE;
END;

BEGIN;
  DISCRETE, 1000, 1, 1;
  RESOURCES: 1, SERVER;
  REPLICATE,1;
END;
```

MODSIM : ModSim é uma linguagem de simulação modular orientada a objetos. Embora inicialmente implementada em processadores *single-threaded*, ele é projetada para uso com computadores paralelos de arquitetura *MIMD - Multiple Instruction Multiple Data* (WEST; MULLARNEY, 1988). A seguir, é apresentado um fragmento de código que representa a uma simulação, utilizando a linguagem *MODSIM*.

```
OBJECT Aircraftobj;

ASK METHOD SetCruise(IN speed: INTEGER);
BEGIN
 BestCruise := speed;
```

```
 IF InFlight
  INTERRUPT SELF FlyDistance;
 END IF;
END METHOD;

TELL METHOD FlyDistance(IN distance; INTEGER);
BEGIN
 InFlight := TRUE;
 WHILE distance > 0.0
  speed := BestCruise;
  start := SimTime;
  WAIT DURATION distance/BestCruise;
   ON INTERRUPT
   elapsed := distance-(eapsed*speed);
 END WAIT;
END WHAILE;
InFlight := FALSE;
OUTPUT("Arrived safely at",SimTime);
END METHOD;
END OBJECT;
```

SIMSCRIPT II : Tem sido amplamente utilizada na programação de simulação de grandes e pequenos modelos, por causa das características da linguagem que facilitam a programação da simulação. A linguagem inclui suporte para simulação de eventos discretos e a combinação de simulação contínua/discreta, gráficos e animações interativas, acesso a banco de dados e um ambiente de desenvolvimento integrado (RICE et al., 2004). O fragmento de código a seguir ilustra o processo de chegada de um cliente em uma fila utilizando a linguagem *SIMSCRIPT*.

```
PROCESS CUSTOMER

 DEFINE TIME.OF.ARRIVAL AS A REAL VARIABLE
 LEFTTIME.OF.ARRIVAL = TIME.V
 REQUEST 1 SERVER(1)
 LET DELAY.IN.QUEUE = TIME.V - TIME.OF.ARRIVAL
 IF NUM.DELAYS = TOT.DELAYS
  ACTIVATE A REPORT NOW
```

```
 ALWAYS
 WORK EXPONENTIAL.F (MEAN.SERVICE.TIME, 2)
 RELINQUISH 1 SERVER(1)
END
```

3.3 *Framework* Para o Desenvolvimento de Programas de Simulação Distribuída

Um *framework* orientado a objetos é um conjunto de classes e interfaces que incorpora um projeto abstrato. Ele provê uma infra-estrutura genérica para construção de aplicações dentro de uma família de problemas semelhantes, de forma que esta infra-estrutura genérica deva 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).

Os *frameworks* são desenvolvidos para prestar um serviço aos desenvolvedores de *software*, pois oferecem soluções prontas ou semi-prontas de problemas específicos já solucionados por outros programadores (CRUZ, 2009).

Existem diversas ferramentas e *frameworks* para o desenvolvimento de aplicações distribuídas, tais como ClassdescM (STANDISH; MADINA, 2004), *Object-Oriented Parallel System* (SONODA; TRAVIESO, 2006), *Object-Oriented Distributed Framework* (DIACONESCU; CONRADI, 2002), SAMRAI–*Structured Adaptive Mesh Refinement Applications Infrastructure* (WISSINK; HYSOM; HORNUNG, 2003), POOMA–*Parallel Object-Oriented Methods and Applications* (DONGARRA et al., 2003), CoSMoS–*Componentbased System Modeler and Simulator*(SARJOUGHIAN; ELAMVAZHUTHI, 2009), WARPP - *A Toolkit for Simulating High-Performance Parallel Scientific Codes* (HAMMOND et al., 2009), SIMCAN – *a SIMulator framework for computer architectures and storage networks* (NUNEZ et al., 2008), *A Simulation Framework for Sensor Networks in J-Sim* (SOBEIH, 2008), iNet – *an extensible framework for simulating immune network* (SUZUKI, 2000), OpenEnergySim – *a 3D internet based experimental framework for integrating traffic simulation and multi-user immersive driving* (NAKASONE et al., 2011), entre outras. Entretanto, estas 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* para o desenvolvimento de aplicações de simulação distribuída. O *framework* de Cruz (2009) tem o objetivo de facilitar o desenvolvimento de programas de

simulação que utilizem uma infra-estrutura distribuída através de uma máquina paralela ou, principalmente, um sistema distribuído.

3.4 *Framework* utilizado

A estrutura oferecida pelo *framework* apresentado por Cruz (2009), por já conter a base para a implementação dos principais mecanismos existentes em um programa de simulação, além de sua alta flexibilidade para adição de novas funcionalidades, foram fatores decisivos para escolha do mesmo na implementação deste trabalho.

É possível, por exemplo, utilizar uma implementação do protocolo *Time Warp*, construído sob as estruturas e funções da biblioteca de troca de mensagens PVM (*Parallel Virtual Machine*) (GEIST et al., 1994), ou uma implementação do protocolo *Rollback* Solidário, utilizando a biblioteca MPI (*Message Passing Interface*) (SNIR et al., 1996). O exemplo de código a seguir, extraído de Moreira et al. (2010), ilustra uma aplicação desenvolvida com os recursos desse *framework*.

```
#include "framework.h"
int main()
{
  Modelo *ModeloParaSimulacao =
    new Modelo ("Modelo.dat");
  Protocolo *TWProtocol =
    new TimeWarp(ModeloParaSimulacao);
  Comunicação *TrocaMensagem = new PVM();
  Arquitetura *SistemaDistribuido =
    new Arquitetura ("Arquitetura.dat");
  Ambiente *AmbienteSimulacao =
    new Ambiente(SistemaDistribuido,
                 TrocaMensagem,
                 TWProtocol);
  AmbienteSimulacao->PrepararAmbiente();
  AmbienteSimulacao->Executar();
  TWProtocol->GerarContabilidade("Saida.dat");
  return 0;
}
```

A seção seguinte apresenta a estrutura básica desse *framework*.

3.4.1 Estrutura do *framework* utilizado

O *framework* em questão é constituído de quatro camadas ou níveis que foram utilizados para o desenvolvimento da ferramenta proposta neste trabalho de mestrado. A figura 3.8 apresenta as divisões em camadas.

Figura 3.8: Arquitetura do *framework* proposto por 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 pelo protocolo que é responsável em realizar a interface entre as camadas de **Comunicação** e **Aplicação**. Neste nível se encontram os protocolos de sincronização para simulação distribuída. Esta camada é responsável em tratar os erros de causa e efeito a fim de garantir a consistência dos resultados obtidos pela simulação.

O quarto e último nível realiza a interface com o usuário, permitindo ao mesmo tempo carregar os modelos a serem simulados e realizar a coleta dos dados após a simulação. Em seu trabalho, Cruz (2009) não tratou desta camada, deixando apenas a especificação de duas classes abstratas para tratamento das questões do modelo que deveriam ser simulados. A especificação desta camada, responsável pela interface com o usuário, é tratada na seção 5.2.

O diagrama de classes da figura 3.9 apresenta a estrutura básica do *framework* proposto por Cruz (2009). A classe `Arquitetura` é a responsável por manter as informações da arquitetura do ambiente distribuído em que o programa de simulação irá executar.

Ela provê uma estrutura para armazenamento das configurações das estações utilizadas, como o número de elementos de processamento, quantidade de memória principal e secundária, protocolos de comunicação, entre outras informações necessárias. Esta classe é fundamental no desenvolvimento deste trabalho, por conter as informações utilizadas pelo ambiente de visualização e monitoramento.

Para as comunicações e trocas de mensagens do programa de simulação desenvolvido com este *framework*, são utilizadas classes especializadas da classe `Comunicacao`. A partir desta classe abstrata, é possível escolher mecanismos específicos que encapsulem as principais bibliotecas de troca de mensagens utilizadas atualmente, tais como MPI (*Message Passing Interface*) (SNIR et al., 1996) e PVM (*Parallel Virtual Machine*) (GEIST et al., 1994).

De forma a garantir a ordem de execução cronológica dos eventos, o *framework* possui a classe abstrata `Protocolo`, cujas classes concretas descendentes oferecem implementação para os mecanismos de sincronização dos processos da simulação. Estas classes implementam os protocolos para simulação distribuída, tais como: *Time Warp*, CMB e *Rollback* Solidário. A classe abstrata `Protocolo` segue o padrão de projeto *Adapter* (GAMMA et al., 2007) sendo o ponto de interação com os componentes existentes nas bibliotecas específicas que tratam do funcionamento de cada mecanismo de sincronização. Por exemplo, para o protocolo *Time Warp*, a especificação pode ser vista no diagrama de classes da figura 3.10, conforme modelagem elaborada por Cruz (2009).

Na figura 3.10, a classe `Processos` é a base do modelo e trata-se de uma classe abstrata, uma vez que a declaração do método `executar()` é apenas um compromisso de que as classes filhas irão implementá-lo. A classe `Estado` representa os atributos de cada processo da simulação. A separação destes atributos da classe `Processo` ocorre devido à necessidade de armazenar os estados durante a simulação, para ser possível realizar o procedimento de *rollback*. O procedimento de salvamento de estados é implementado nas especializações da classe `Checkpoint`, como, neste caso, o SSS – *Sparse State Saving* (FLEISCHMANN; WILSEY, 1995). O diagrama apresenta duas classes ativas: `Emissor` e `Receptor`, que são responsáveis pela comunicação com as rotinas do nível 1 da arquitetura em camadas do ambiente de simulação (figura 3.8). A classe `Observador` tem a função de permitir a implementação de mecanismos para troca dinâmica de protocolos ou para migração de processos. É através das especializações desta classe que será possível implementar o mecanismo de monitoramento dos processos da simulação.

Em adição, a modelagem do protocolo *Rollback* Solidário, com os mesmos propósitos da modelagem apresentada para o protocolo *Time Warp*, pode ser encontrada em Moreira (2005).

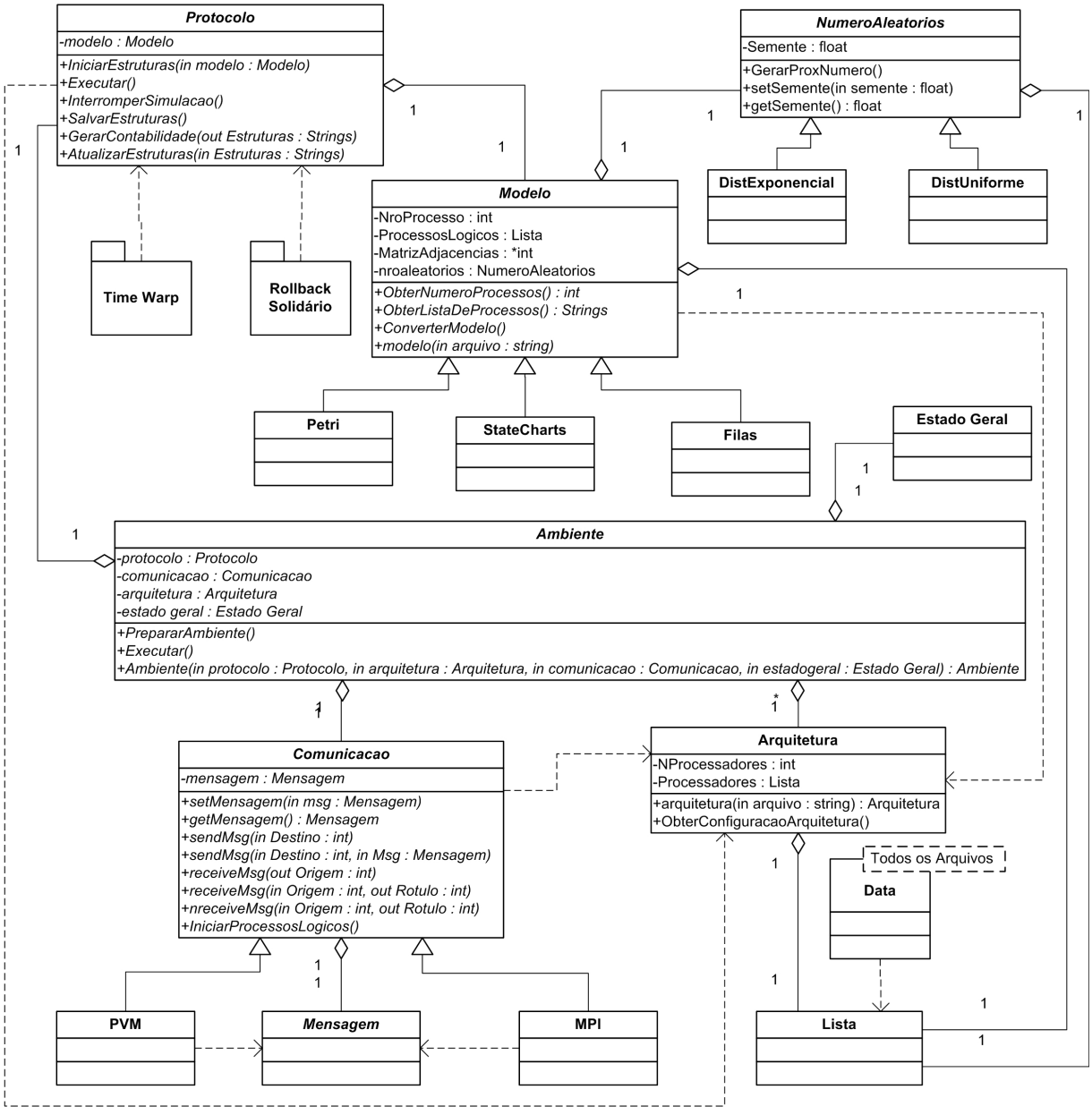

Figura 3.9: Diagrama de classes do *framework* proposto por Cruz (2009)

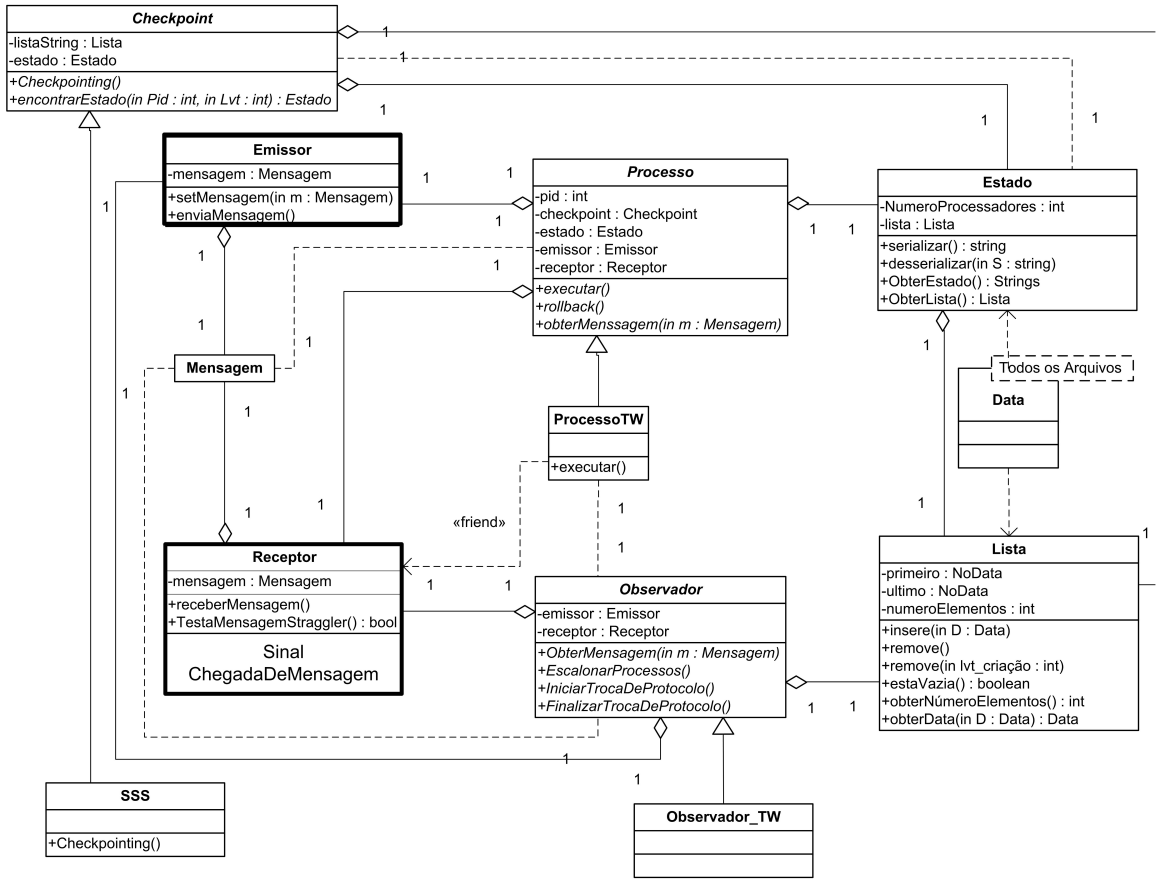

Figura 3.10: Diagrama de classes do protocolo *Time Warp* (CRUZ, 2009)

3.5 Considerações finais

A *simulação* vem ganhando cada vez mais adeptos, devido a sua capacidade de conseguir prever o comportamento de sistemas, através de modelos que os representam nas mais variadas situações hipotéticas, sem colocar em risco o sistema real e servindo como uma ferramenta para auxílio nas tomadas de decisões e análise de desempenho. Devido aos benefícios e a grande ascensão de usuários da simulação, atualmente existem dezenas de ferramentas comerciais voltadas para este tipo de aplicação nas mais diversas áreas.

A complexidade existente no desenvolvimento da simulação em um ambiente distribuído, devido aos problemas inerentes a *computação paralela*, pode ser contornada através do desenvolvimento de uma ferramenta com a utilização de *frameworks* específicos, que se encarreguem da execução distribuída. Com a utilização da simulação distribuída, é possível simular sistemas maiores e mais complexos em um menor espaço de tempo, diferente do que acontece no cenário sequencial.

Por sua vez, utilização de *frameworks* no desenvolvimento de um *software* pode trazer grandes benefícios, como, por exemplo, a diminuição do tempo de desenvolvimento de certas rotinas que podem ser obtidas a partir de um *framework* que as implementa.

O *framework* proposto por Cruz (2009) desempenha um importante papel no desenvolvimento da ferramenta proposta neste trabalho de mestrado, pois oferece componentes específicos para o desenvolvimento de programas de simulação distribuídos usando a abordagem SRIP.

Capítulo 4

Monitoramento de Processos na Simulação Distribuída

O escalonamento de processos é uma atividade computacional que tem grande influência no desempenho de um sistema computacional distribuído, pois uma boa distribuição dos processos entre os processadores disponíveis é indispensável para que se obtenha um bom desempenho (SOUZA et al., 2008). Ele consiste, basicamente, na alocação de recursos computacionais para a execução de um programa formado por vários processos. Em um ambiente distribuído é necessário o uso de um algoritmo ou política de escalonamento para definir quais recursos e por quanto tempo serão alocados para cada processo. Este algoritmo deve ser projetado de acordo com requisitos diferentes e, frequentemente, deve atender a requisitos opostos entre si o que dificulta a escolha por uma política adequada (JUNQUEIRA, 2012).

Um programa de simulação distribuída é afetado pela máquina paralela e pelas características do próprio modelo de simulação. Existem várias técnicas para otimizar o desempenho de um programa de simulação distribuída. As técnicas que mais se destacam são as de escalonamento e de balanceamento de carga (CAROTHERS; FUJIMOTO, 2000).

Apesar dos termos "escalonamento de processos" e "balanceamento de carga" frequentemente serem empregados como sinônimos, o escalonamento de processos deve ser visto como uma atividade que tem como objetivo o balanceamento da carga entre os vários elementos de processamento. O balanceamento de carga tenta distribuir uniformemente os recursos compartilhados, evitando a situação em que um elemento de processamento está carregado enquanto que outro está com a carga de processamento leve (BRANCO, 2004).

Segundo Carothers e Fujimoto (2000), algumas causas do desbalanceamento de carga em uma simulação distribuída, baseada no protocolo *Time Warp*, são:

Avanço não homogêneo dos relógios lógicos dos processos: os processos da simulação podem estar executando em processadores com diferentes cargas e esta diferença pode levar ao avanço mais rápido de alguns processos da simulação. Os processos que se atrasam podem gerar *rollbacks* longos e frequentes;

Influências de outras aplicações: outras aplicações que concorrem com o uso dos recursos de *hardware* podem afetar o balanceamento de carga de um programa de simulação;

Fatores internos da simulação: a diferença de parâmetros entre os processos da simulação pode levar ao desbalanceamento de carga na simulação.

O monitoramento dos processos da simulação é essencial para que se obtenham dados necessários à tomada de decisão a respeito do escalonamento dos processos. A utilização de informações da simulação distribuída pode levar a um melhor desempenho quando é empregada para determinar o escalonamento de processos. Uma vez determinada uma situação de desbalanceamento, a migração de processos é utilizada para recuperar o equilíbrio na carga do sistema. Neste contexto, Junqueira (2012) propôs dois mecanismos para realizar a migração de processos em uma aplicação de simulação distribuída baseada no protocolo *Time Warp*. Ambos os mecanismos são capazes de obter informações da simulação e disponibilizá-los para um algoritmo de balanceamento. Um dos mecanismos é o de migração coletiva, que consiste no término e na recriação de todos os processos da simulação. O outro é o mecanismo de migração individual, no qual apenas alguns processos são encerrados e recriados. Os resultados deste trabalho demonstraram que o impacto da migração é pequeno na simulação e, desta forma, utilizado com um algoritmo adequado de balanceamento pode-se aumentar o desempenho da simulação distribuída, particularmente em sistemas heterogêneos, como, por exemplo, na computação em nuvens.

Por conseguinte, um programa de simulação distribuído pode tirar grande proveito de um sistema de monitoramento. Informações como o estado dos processos lógicos, quantidade de *rollbacks* realizados, processos ativos ou em migração, dados sobre o funcionamento da rede de interconexão, entre outras, podem ser usadas para múltiplas tarefas, incluindo: detecção de falhas, escalonamento, depuração e análise dos resultados parciais da simulação. Além disso, é possível armazenar um histórico das execuções do programa, a fim de analisar o comportamento do sistema ao longo de um determinado período de tempo ou utilizar esses dados nos algoritmos de mapeamento dos processos da simulação, para que, em uma nova execução, o mapeamento possa ser melhor definido, diminuindo a necessidade de migrações durante a simulação.

4.1 Etapas do monitoramento de um sistema distribuído

Segundo Tesser (2011), um sistema de monitoramento deve compreender funcionalidades relacionadas à geração de dados de monitoramento, seu processamento, distribuição e apresentação. A ordem em que as tarefas de cada tipo são realizadas pode variar. Por exemplo, pode ser realizada alguma forma de processamento dos dados tanto antes quanto após sua distribuição, assim como em ambos desses momentos. Tudo isso deve ser projetado e implementado levando em conta que existem vários problemas associados ao monitoramento de sistemas distribuídos. Um deles é a possibilidade de que os dados gravados não sejam suficientemente atuais, devido ao atraso na transmissão dos mesmos. Outro problema é que os eventos não são registrados na ordem de ocorrência, o que torna necessária a utilização de alguma técnica de sincronização. A necessidade de filtrar e processar estas informações é outra dificuldade, uma vez que o sistema de monitoramento pode competir pelo uso de recursos com o sistema sendo observado, modificando o comportamento normal do sistema.

4.1.1 Geração dos dados de monitoramento

Tesser (2011) define um evento como uma representação das mudanças de estados que ocorrem em um determinado instante em cada uma das componentes que compõem o sistema a ser monitorado, tornando-as observáveis. A obtenção desses dados pode ser realizada através de um relatório periódico em que o envio desses dados é feito sempre ao fim de um ciclo de tempo. Outra maneira é fazer com que os dados sejam enviados sob demanda.

Em um sistema distribuído, como os recursos observados estão distribuídos, os dados de monitoramento também estão. Portanto, há necessidade de um mecanismo de coleta, que seja capaz de reunir as informações obtidas. As transmissões de dados entre dois componentes de um sistema podem ser feitas apenas quando requisitadas pelo receptor. Esse modelo é chamado de *pull* ou sob demanda. No caso contrário, quando o remetente é quem decide quando enviar os dados, sem necessidade de requisição pelo receptor, o modelo é chamado *push* (TESSER, 2011). A utilização de três tipos de componentes, chamados de coletores, agregadores e clientes, possibilita a coleta dos dados de monitoramento distribuído.

A utilização dos rastros de monitoramento tem como objetivo armazenar de forma organizada, a partir da ordem de ocorrência, os eventos e informações de componentes durante um intervalo de tempo, para que sejam utilizados posteriormente em alguma

tarefa que os necessitem. Algumas dessas tarefas são: arquivamento e análise *post-mortem*, verificação de problemas relacionados com a escassez de recursos disponíveis e manutenção da visão lógica das atividades do sistema, que permite a criação de diferentes rastros globais a partir de rastros locais.

4.1.2 Processamento dos dados de monitoramento

Após a coleta dos dados requisitados, faz-se necessário o processamento desses dados. O processamento pode ocorrer por vários propósitos diferentes, seja a integração de múltiplos rastros, a geração de rastros específicos a partir de um mais abrangente, a validação dos dados, sua filtragem, a combinação ou correlação dos mesmos e a sua análise. A utilização de mecanismos para verificação da corretude dos dados é um processamento executado nessa fase. Como visto em Tesser (2011), outra importante atividade que ocorre durante o processamento é a filtragem dos dados, que tem como objetivo o seu tratamento de modo a diminuir o seu volume através de abstrações que mantém somente os dados relevantes para a análise. A utilização de filtragens pode ocasionar uma diminuição no consumo de CPU, assim como também de banda de comunicação.

4.1.3 Distribuição de informações de monitoramento

A distribuição de informações de monitoramento consiste na utilização de mecanismos que possibilitem que a obtenção das informações adquiridas pelos agentes locais possam ser entregues a um cliente específico. Para isso, os mecanismos mais simples são a comunicação por *broadcast* ou *multicast* ou, ainda, em mecanismos mais complexos de publicação e assinatura de informações. Neste contexto, haverá um serviço de eventos que mantém um banco de dados de eventos publicados e assinantes interessados na publicação. Eventos em um objeto de interesse são publicados no serviço de eventos. Assinantes informam ao serviço de eventos a respeito dos tipos de eventos que eles estão interessados. Quando um evento ocorre no objeto de interesse uma notificação é enviada para os assinantes daquele tipo de evento (COULOURIS; DOLLIMORE; KINDBERG, 2001).

4.1.4 Apresentação de informações de monitoramento

Considerado a etapa final do processo de monitoramento, a apresentação consiste em disponibilizar os dados obtidos durante a monitoração para seus devidos fins, podendo ser realizada através de representações gráficas a partir de diagramas, animações gráficas ou textual e também de acordo com um padrão predeterminado. Desta forma, pode-se utilizar esses dados juntamente com algum algoritmo de balanceamento de carga.

4.2 Problemas inerentes ao monitoramento distribuído

O monitoramento é uma atividade complexa, devido a inexistência de um relógio global de referência nos sistemas distribuídos. Desta forma, por causa dos atrasos arbitários na transmissão de mensagens, é impossível obter um *snapshot* instantâneo, o que dificulta a construção de um estado global consistente do sistema para a tomada de decisões. Por conseguinte, vários problemas podem ocorrer.

A monitoração de um sistema pode acabar afetando de alguma forma o comportamento do sistema, por exemplo, pode diminuir o seu desempenho, mudar a ordem global dos eventos e até mesmo produzir resultados incorretos. Diferentes execuções de um mesmo algoritmo, em um sistemas paralelo ou distribuído, podem resultar em diferentes intercalações de eventos. Isso faz com que o comportamento do sistema seja não-determinístico e imprevisível. Como resultado deste não determinismo, há um fenômeno conhecido como efeito de intrusão, que resulta do fato da observação de um programa poder afetar o comportamento do mesmo, ou seja, a tentativa de ganhar mais informações sobre o programa pode contribuir para a alteração do seu comportamento. Desta forma, é importante que o mecanismo de monitoramento cause a menor intrusão possível no sistema, minimizando assim a ocorrência dos problemas causados pela concorrência com os processos de monitoramento.

A troca de mensagens contendo *timestamps* com o uso de relógio lógicos vetoriais, são soluções que podem ser utilizadas para contornar estes problemas (BABAOGLU; MARZULLO, 1993).

4.3 Considerações Finais

Como visto no início deste capítulo, as principais causas do desbalanceamento de carga na simulação distribuída podem gerar uma diminuição no desempenho da simulação e, consequentemente, levar mais tempo para sua conclusão. Através dos mecanismos de monitoramento apresentados, é possível acompanhar o que ocorre em cada nó do sistema durante a execução do programa de simulação. Desse modo, é possível contabilizar a ocorrência de eventos que, de certa forma, ocasionem uma diminuição de desempenho, como por exemplo, a quantidade de *rollbacks* ocorridos.

Essas informações, aquisitadas a partir do monitoramento, possibilitam que os algoritmos específicos para o balanceamento de carga ajam de forma a melhorar o desempenho na execução da simulação através da realocação dos processos nas estações de processamento, ou seja, da migração dos processos.

Capítulo 5

Projeto da Ferramenta

Este capítulo apresenta o projeto da ferramenta para simulação de eventos discretos, proposta neste trabalho de mestrado. Trata-se de um *software* paralelo, ou seja, a simulação ocorrerá em uma plataforma computacional distribuída, conforme discussão realizada no capítulo 2. A principal característica, que difere esta ferramenta de outras ferramentas paralelas de simulação disponíveis, é a capacidade de configurar ou selecionar protocolos de simulação, bibliotecas de troca de mensagens e a infraestrutura computacional, possibilitando, ainda, o monitoramento dinâmico dos processos da simulação.

Neste contexto, o objetivo é criar uma ferramenta que, além de prover recursos para melhorar o desempenho da simulação, auxilie um usuário de simulação, mas iniciante em ambientes computacionais distribuídos, a entender o funcionamento da arquitetura e conseguir utilizar, de uma maneira eficiente, os recursos que este tipo de plataforma pode oferecer. Por sua vez, o monitoramento, aliado a mecanismos de escalonamento, permite automatizar o mapeamento dinâmico de processos, favorecendo a simulação de modelos complexos desenvolvidos por usuários experientes.

As características especiais desta aplicação, para um bom projeto, exigem a separação dos seus diversos componentes em camadas. Particularmente, estas características podem ser agrupadas em 3 categorias, como visto na figura 5.1:

1. as questões relacionadas com computação paralela/distribuída, envolvendo os problemas inerentes ao escalonamento de processos, consistência dos dados, erros de causalidade na sincronização dos processos e, em particular, os problemas envolvendo os protocolos de sincronização para simulação distribuída e o particionamento do modelo;

2. as questões relacionadas à aplicação em si, ou seja, modelagem, simulação e análise dos resultados;

Figura 5.1: Principais características da Aplicação.

3. a integração do programa de simulação em um ambiente distribuído com a interface de comunicação com o usuário da aplicação.

Por conseguinte, para facilitar o tratamento e a integração destas partes, o projeto da ferramenta está estruturado conforme a arquitetura de *software* em camadas. O objetivo desta arquitetura é separar as tarefas de acordo com suas pertinências. A estruturação da ferramenta segue o modelo em camadas definido para o *framework* desenvolvido no trabalho de Cruz (2009). Como comentado no capítulo 3, este *framework* tem o objetivo de facilitar o desenvolvimento de programas de simulação que utilizem uma infraestrutura distribuída através de uma máquina paralela ou, principalmente, de um sistema distribuído.

5.1 Extensão do framework

Como dito anteriormente, para o desenvolvimento deste trabalho foi adotado o *framework* criado por Cruz (2009), este *framework* foi estruturado em camadas para alcançar um alto nível de flexibilidade e usabilidade. Entretanto, não foram previstos recursos para se realizar o monitoramento dos processos da simulação. A classe `Observador` pode realizar atividades de gerenciamento local e foi construída para permitir a implementação de mecanismos para troca dinâmica de processos e, eventualmente, migração de processos, mas sua especificação diz respeito a visão local de um processo da aplicação e não da visão global de um determinado sistema.

Neste contexto, para permitir que a ferramenta proposta neste trabalho de mestrado forneça recursos de monitoramento, e, com isso, permita a implementação de mecanismos de escalonamento dinâmico dos processos da simulação, novas classes foram especificadas, expandindo os diagramas apresentados nas figuras 3.9 e 3.10.

A figura 5.2 apresenta três novas classes e suas relações com as demais classes do

framework. São classes abstratas para permitir que algoritmos eficientes de escalanomento e migração de processos possam ser implementados e/ou substituídos futuramente, sem danos para a aplicação. A classe `Monitor` é a principal e tem a função de integrar os recursos necessários para realizar o monitoramento da simulação e, se necessário, tomar as decisões referentes ao escalonamento dos processos. Seus recursos se assemelham àqueles presentes na classe `Observador`, tendo o mesmo relacionamento com as classes ativas `Emissor` e `Receptor`. O método `ConstruirEstadoGlobalConsistente()` é o mais importante. Este método é abstrato e deve ser implementado nas especializações desta classe. O estado global obtido em uma aplicação distribuída pode ser inconsistente ou desatualizado, uma vez que não existe um relógio global de referência. Neste contexto, para ser obtido um estado global consistente, técnicas específicas devem ser utilizadas, como pode ser conferido no trabalho de Babaoglu e Marzullo (1993). A relação da classe `Monitor` com a classe `Arquitetura`, disponível no *framework*, se refere a necessidade de obter informações dos elementos de processamento para uma eventual necessidade de reconfigurar o mapeamento dos processos da simulação. Também é importante este relacionamento para possibilitar a implementação de um mecanismo para troca de protocolos ou migração dos processos, razão pela qual a classe `Monitor` também mantém um relacionamento com a classe `Observador`.

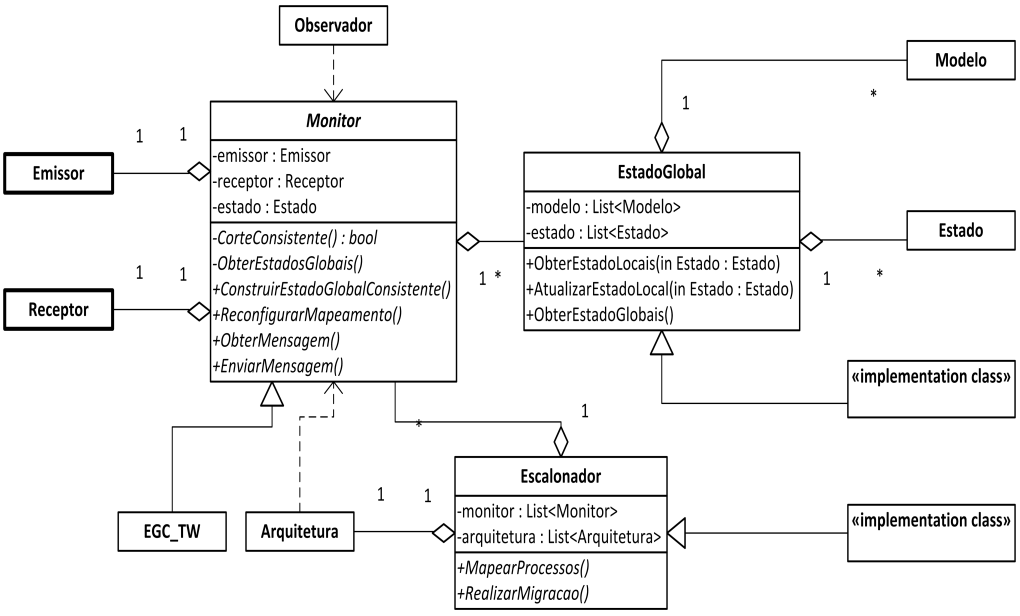

Figura 5.2: Diagrama de classes para o monitoramento de processos

Para se ter uma fotografia completa de uma aplicação distribuída é preciso obter o estado de cada processo que a compõe. Para este fim, a classe `EstadoGlobal` mantém uma lista atualizada dos estados locais de cada processo lógico. A lista é formada por objetos da classe `Estado`, disponível no *framework*. Da mesma forma, a classe `Modelo` faz parte da composição da classe `EstadoGlobal`, visto que as informações da quantidade de processos já se encontram disponíveis naquela classe.

O diagrama de sequência da figura 5.3 ilustra o comportamento de um objeto da classe `Monitor` ao receber a mensagem `ConstruirEstadoGlobalConsistente()`. Após o objeto `Monitor` receber esta mensagem, ele inicia o método `ObterEstadosGlobais()` que, por sua vez, irá obter os estados locais de cada processo da simulação. Quando todos os processos locais receberem as respectivas mensagens de solicitação do estado local e retornarem o resultado, será verificado se o corte correspondente aos estados locais é consistente, sinalizando um estado global consistente (BABAOGLU; MARZULLO, 1993). Caso contrário, o algoritmo deverá atualizar as informações, até que um corte consistente esteja disponível.

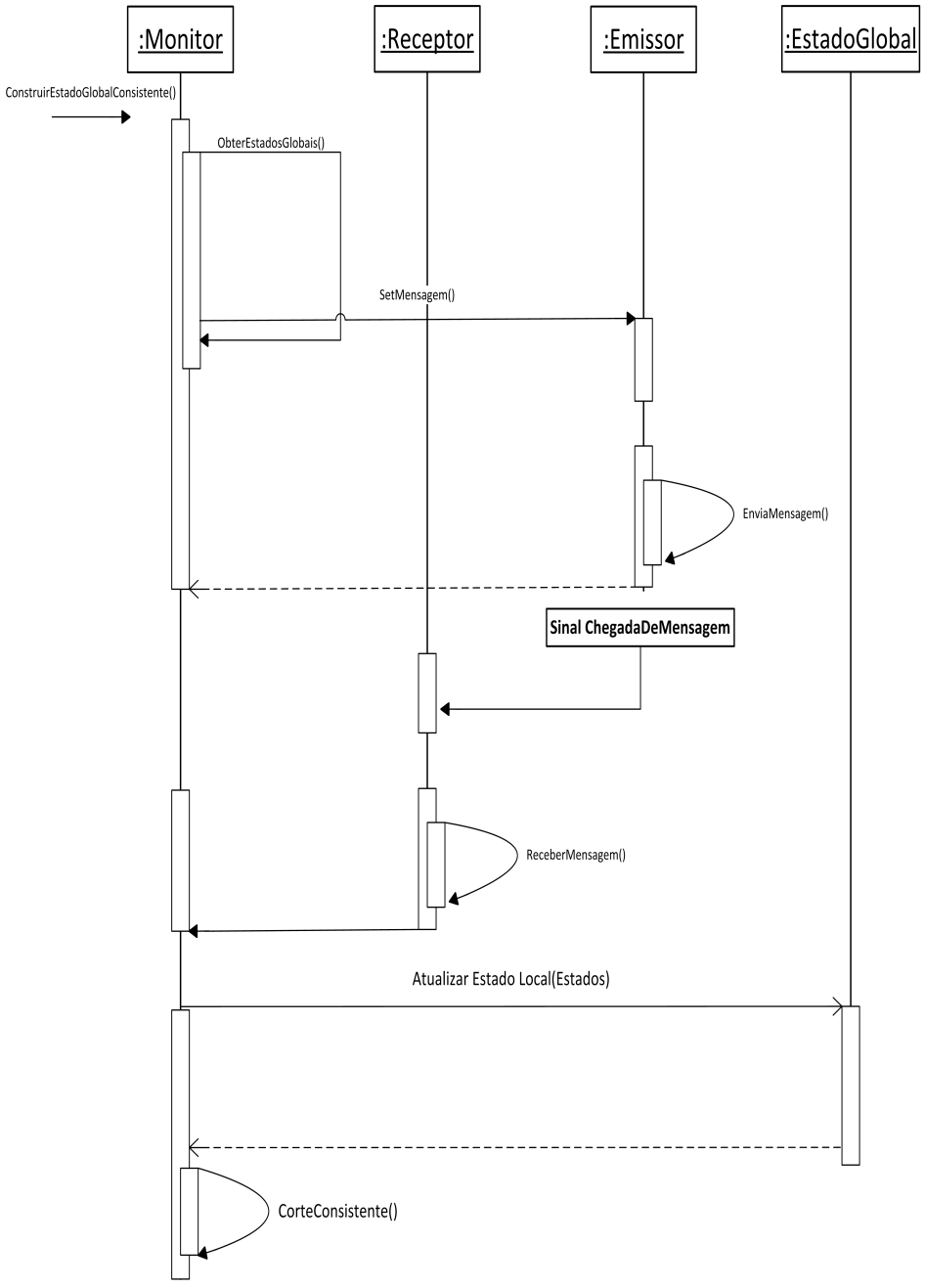

Figura 5.3: Diagrama de sequência para o método `ConstruirEstadoGlobalConsistente`

Por fim, a classe `Escalonador` possui as funcionalidades necessárias para implementar as políticas de escalonamento e mapear os processos de forma estática ou dinâmica, permitindo realizar a migração dos processos para manter o balanceamento de carga da simulação. O dois métodos principais desta classe, `MapearProcessos()` e `RealizarMigracao()`, são responsáveis por definir o escalonamento dos processos da simulação. O método `MapearProcessos()` será utilizado para produzir um novo mapeamento, caso já tenha havido um mapeamento anterior. Neste caso, uma nova organização será utilizada para provocar a migração dos processos, que será realizada através da chamada ao método `RealizarMigracao()`.

O trabalho de mestrado de Junqueira (2012) apresenta um estudo aprofundado sobre migração de processos na simulação distribuída e propõe dois mecanismos para realizar essa migração. Os dois mecanismos serão utilizados na ferramenta projetada neste trabalho de mestrado através da especialização da classe `Escalonador` e implementação dos métodos abstratos `MapearProcessos()` e `RealizarMigracao()`.

5.2 Interface gráfica

Um bom tratamento visual de um *software* permite que as suas funcionalidades sejam melhor assimiladas por parte do usuário, facilitando a sua utilização. Na arquitetura em camadas da figura 3.8, a camada **Aplicação** tem como uma de suas atribuições apresentar os dados da aplicação.

As interfaces com o usuário são construídas a partir de componentes de interface individuais: botões, campos de texto, réguas de cálculo etc. Esses componentes, por sua vez, são distribuídos em componentes tipo *contêineres*, como, por exemplo, formulários e painéis. Neste contexto, o projeto desta camada visa oferecer ao usuário um ambiente que possibilite criar, executar e monitorar a simulação, de forma ágil, fácil e amigável.

A modelagem apresentada nesta seção irá destacar apenas a estrutura básica definida para a construção dos objetos da ferramenta e como eles se relacionam. As classes utilizadas para construção de componentes tradicionais não serão descritas, uma vez que serão utilizadas bibliotecas conhecidas, em particular, a biblioteca *Swing* da linguagem Java. Os efeitos visuais desta estruturação podem ser verificados no próximo capítulo, quando será apresentado o projeto da interface, através da descrição da implementação do primeiro protótipo da ferramenta.

A figura 5.4 apresenta o diagrama das classes que foram adicionadas ao *framework* para a construção da interface gráfica da ferramenta proposta. A classe abstrata `ComponenteVisual` é a classe mãe de todos os componentes visuais da ferramenta, fornecendo os recursos necessários para as especializações das demais classes utilizadas para represen-

tação gráfica de componentes. Como exemplo, cada elemento que compõe as linguagens simbólicas utilizadas para a modelagem de um sistema tem sua classe pertinente.

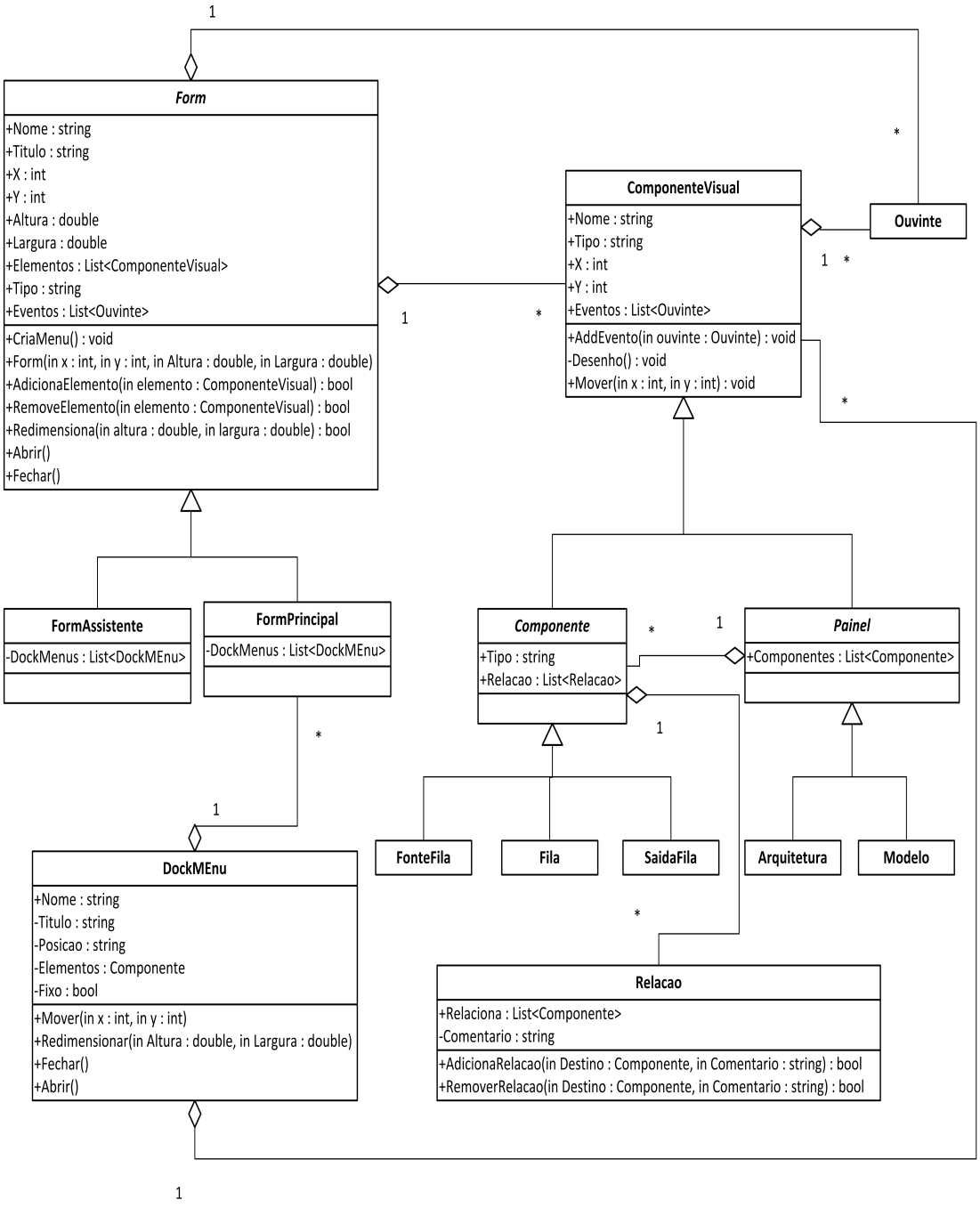

Figura 5.4: Diagrama de classes da camada Visual

Um importante mecanismo está presente na classe `ComponenteVisual`, chamado ouvinte. A classe `Ouvinte` fornece aos componentes uma interface que permite a comunicação entre as demais classes das outras camadas. Como exemplo, pode-se considerar a notificação que durante um evento de criação de um modelo, o usuário reposicionou um objeto de fila. Neste caso, a classe `Fila` notifica, através da classe `Ouvinte`, que tal objeto deve ser redesenhado na nova posição. A classe `ComponenteVisual` contém uma lista de

`Ouvintes`, que são herdados para suas classes filhas.

A classe `Componente` é uma classe abstrata intermediária que adquire uma herança da sua classe base. Sua função é fornecer uma base para a especialização das demais classes que representam componentes manipuláveis do ambiente, como, por exemplo, `SaidaFila`, classe usada para representação gráfica do fim do atendimento de uma fila. Nesta classe, o atributo `Tipo` serve para especificar o tipo de linguagem simbólica que o objeto pertece. Existe também uma agregação da classe `Relacao`, que permite expressar com quais outros objetos o objeto principal se relaciona.

Assim como a classe `Componente`, a classe `Painel` também é uma classe abstrata intermediária, responsável por fornecer os recursos básicos para suas especializações. A lista de componentes existentes nesta classe permite que suas classes filhas recebam qualquer um dos componentes implementados no ambiente. As classes filhas da classe `Painel` podem se especializar nas seguintes classes: `Modelo`, utilizada para manipulação visual dos modelos criados, ou `Arquitetura`, que representa a estrutura do ambiente distribuído onde ocorre a simulação.

A classe abstrata `Form` fornece a base de criação para as janelas do ambiente. A janela principal é formada pela classe `FormPrincipal`, que é uma classe herdada da classe `Form`. Em sua especialização, a classe `FormPrincipal` recebe o atributo `DockMenus`, que possibilita a criação de menus manipuláveis, que podem ser fixados nas laterais da janela principal, ou até mesmo serem fixados dentro de outros menus. Outra possibilidade desses menus, é o redimensionamento e a possibilidade de manipulação flutuante, que permite posicionar o menu sobre qualquer área da janela principal.

A classe `Form` ainda pode se especializar em outro tipo de formulário, como, por exemplo, `FormAssistente`, utilizado para criar assistentes passo a passo que auxiliem os usuários na construção de novos projetos ou arquivos.

5.3 Considerações finais

A arquitetura em camadas, utilizada no desenvolvimento desta ferramenta, permitiu a separação das diferentes estruturas empregadas no projeto. Cada camada executa funções bem definidas, facilitando a tarefa de desenvolvimento e também de manutenções futuras do *software*. A adoção dessa arquitetura também visa o reaproveitamento de código.

A adoção do *framework* proposto por Cruz (2009), juntamente com os métodos de migração de processos propostos por Junqueira (2012), facilitou o desenvolvimento da ferramenta, pois ele fornece grande parte dos recursos necessários para a troca de mensagens e para o sincronismo dos processos da simulação, além das classes necessárias à

execução de um programa de simulação.

Capítulo 6

Primeiro Protótipo da Ferramenta

A utilização de um ambiente gráfico para simulação distribuída tem como principal objetivo facilitar o processo de criação, execução e monitoramento de uma simulação. Com o uso de representações gráficas, por exemplo ícones, que representam de forma visual um componente do sistema, é possível criar facilmente um modelo, que posteriormente será utilizado para ser simulado. No âmbito da simulação, é possível criar animações sobre o modelo para demonstrar a evolução da simulação, dando ao usuário uma melhor interface para visualizar os resultados.

O módulo de modelagem deve prover componentes que possibilitem ao usuário criar modelos confiáveis de maneira rápida e fácil utilizando uma das linguagens simbólicas (Redes de Petri, Redes de Fila e *Statecharts*) presentes na ferramenta.

Procurando reduzir o tempo de execução de grandes simulações, o uso da computação distribuída permite a simulação de grandes sistemas que, em um ambiente sequencial, levariam horas para ser finalizado. Em uma abordagem distribuída, a simulação está sujeita aos problemas inerentes a este tipo de arquitetura computacional, por exemplo, erros de causalidade. A ferramenta em questão deve ser capaz de detectar ou corrigir a ocorrência desses eventos automaticamente através dos protocolos de sincronização disponíveis.

Como um diferencial entre as outras ferramentas de auxílio ao desenvolvimento de simulações, a funcionalidade de monitoramento da arquitetura permite ao usuário acompanhar a evolução da sua simulação sobre o sistema computacional utilizado.

Este capítulo apresenta uma discussão sobre o desenvolvimento do primeiro protótipo da ferramenta. O trabalho se concentrou no desenvolvimento da interface gráfica, uma vez que existem outros pesquisadores do Grupo de Engenharia de Sistemas e de Computação (GPESC) da UNIFEI trabalhando no aprimoramento do *framework* e, em especial, na implementação dos protocolos *Time Warp* e *Rollback* Solidário. Destaca-se que o objetivo

na criação de um protótipo é demonstrar a aplicabilidade do projeto.

O projeto gráfico da interface foi desenvolvido utilizando a Linguagem C#, devido a facilidade para a montagem dos formulários e definição dos componentes visuais. O objetivo, neste caso, é verificar as questões de usabilidade antes da implementação de uma versão completa da ferramenta. Entretanto, a versão final está sendo construída utilizando a Linguagem Java, por ser, principalmente, uma linguagem multiplataforma com recursos que facilitam a construção de mecanismos dinâmicos para a migração dos processos e por ser uma das bases da implementação das classes dos protocolos *Time Warp* e *Rollback* Solidário.

6.1 Funcionalidades

Usabilidade é um termo usado para definir a facilidade com que um usuário utiliza uma ferramenta ou objeto a fim de realizar uma determinada tarefa. Buscando oferecer um ambiente de fácil adaptação ao usuário, foi adotada uma organização das funcionalidades operacionais da ferramenta em abas, permitindo o acesso fácil aos módulos e configurações desejados, sendo a interface composta pelas, seguintes funcionalidades:

- *sandBox* - localizada na parte central da ferramenta, trata-se de uma área onde ocorre todas as ações de modelagem, execução e monitoramento da simulação;
- *Ferramentas* - aba lateral superior esquerda que abriga todos os componentes disponíveis para a modelagem de um sistema. Em seu interior os componentes estão organizados de acordo com as seguintes categorias: Fila, Redes de Petri, *Statecharts*, fornecendo os elementos essenciais para a modelagem utilizando essas linguagens simbólicas;
- *Configurações* - nesta aba lateral inferior esquerda é possível configurar os protocolos do sistema distribuído, permitindo a escolha do protocolo de troca de mensagens MPI (*Message Passing Interface*) ou PVM (*Parallel Virtual Machine*), a escolha do protocolo de sincronismo *Time Warp* e *Rollback* Solidário e a adição de novos protocolos de sincronização;
- *Propriedades* - localizada na lateral inferior direita, esta aba abriga os atributos de configuração pertinentes ao componente selecionado que esteja na *sandBox*, permitindo assim a alteração dos mesmos;
- *Projeto* - nesta aba lateral superior direita estão disponíveis as opções responsáveis pela estrutura da simulação, armazenando de forma estruturada os arquivos que compõem o projeto de simulação trabalhado;

- *Barra de menu principal* - esta barra de menu superior contém as funcionalidades padrões de uma aplicação, como, exemplo, menu arquivo, menu editar, menu ferramentas e o menu ajuda;

- *Barra de menu ferramentas* - localizada logo abaixo do menu principal, abriga as funcionalidades para criar um novo projeto de simulação, abrir um projeto existente, salvar o projeto ativo, iniciar uma simulação e parar a simulação em operação.

A disposição visual permite que o usuário tenha, de forma organizada, em um único plano, todos os recursos necessários para elaborar sua simulação, evitando assim o grande fluxo de navegação por menus ou janelas, assim visto na imagem 6.1. Os componentes localizados na caixa de ferramentas podem ser usados através de um *clique* do *mouse* e, posteriormente, serem arrastados até a posição desejada sobre o *sandBox*. A configuração dos atributos pode ser acessada através da seleção do componente desejado, permitindo a modificação dos atributos do objeto através da aba *Propriedades*.

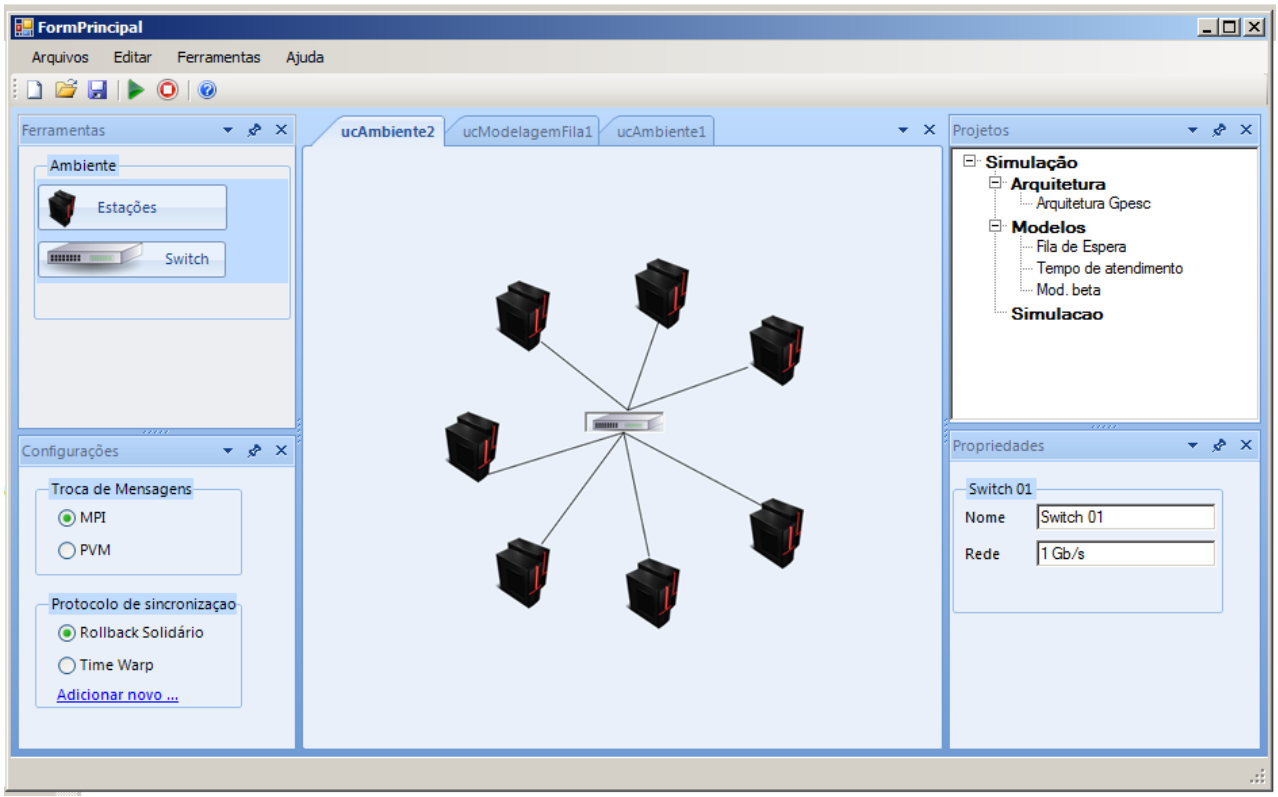

Figura 6.1: Imagem geral do protótipo desenvolvido.

6.1.1 *sandBox*

O conceito de *sandBox* adotado nesta ferramenta é uma analogia as caixas de areia existentes na grande maioria dos parques e escolas, onde crianças têm a liberdade e a

facilidade de criar e destruir modelos. O *sandBox*, apresentado na figura 6.2 e adotado na ferramenta, desempenha um papel semelhante a caixa de areia do parque, servindo como um ambiente que possibilite a criação de modelos dos mais variados sistemas.

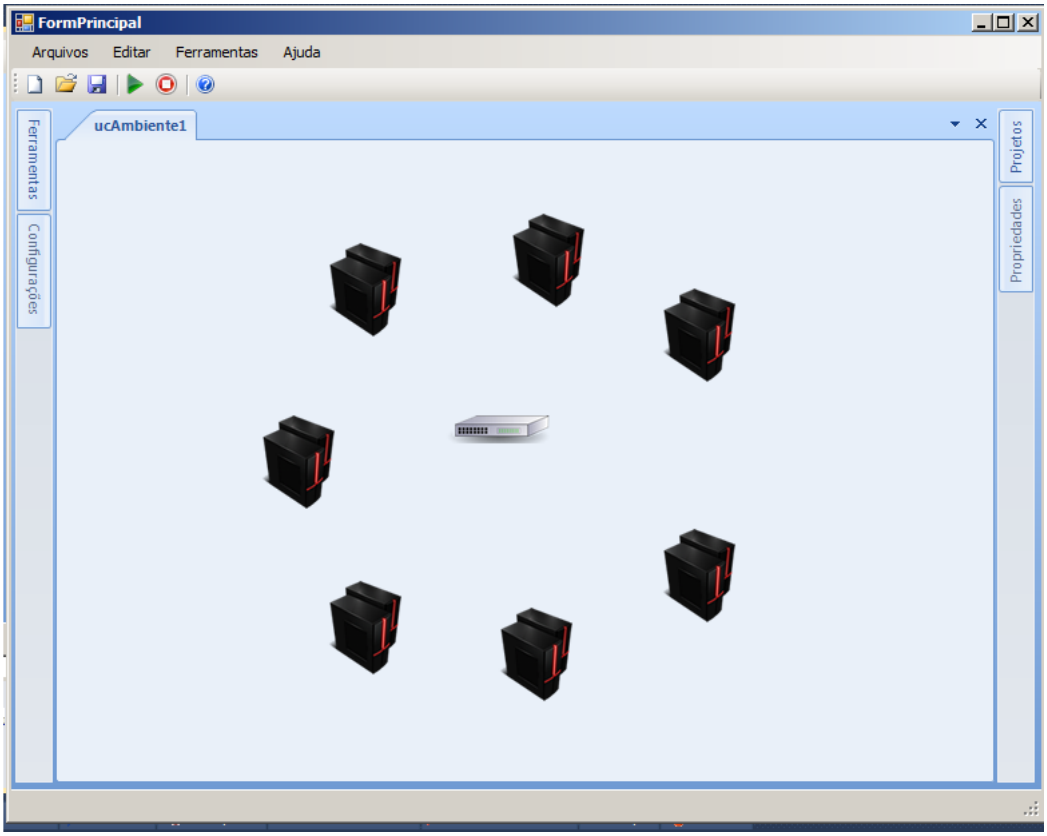

Figura 6.2: Destaque para a interface central denominada *sandBox*

Através deste componente, o usuário tem total controle sobre a manipulação dos demais componentes que se encontram sobre o *sandBox*, desde a configuração dos atributos, reposicionamento em qualquer localização delimitada do *sandBox*, replicação dos componentes e, também, a exclusão dos mesmos.

6.1.2 Aba Ferramentas

A caixa de ferramentas (figura 6.3), localizada na parte esquerda superior do ambiente, abriga os principais componentes organizados de acordo com a linguagem simbólica escolhida para a criação de um modelo sobre o componente *sandBox*. Desta forma, caso seja escolhida *Redes de Fila*, como linguagem simbólica para a criação do modelo, somente serão exibidos os componentes associados a esta ferramenta de modelagem.

A utilização de um componente que se encontra na aba de ferramentas ocorre a partir de um único *clique* sobre o componente escolhido e um arrasto até a posição desejada sobre o *sandBox*.

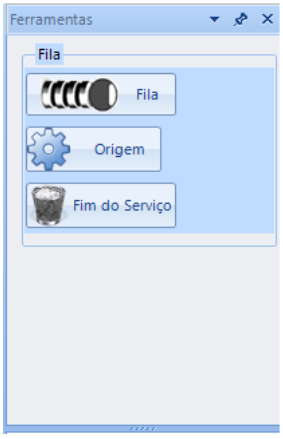

Figura 6.3: Aba de Ferramentas contendo os componentes referentes a Redes de Fila.

A caixa de ferramentas, assim como todas as demais abas, pode ser movimentada para qualquer posição do ambiente, através de um *clique* em seu cabeçalho e arrasto para a posição desejada. Outros controles estão presentes no cabeçalho como, por exemplo, fixação da aba e fechamento da aba, funcionalidades também aplicadas as demais abas existentes no ambiente.

6.1.3 Aba Configuração

A caixa de configuração permite que seja escolhido qual protocolo será utilizado durante a simulação (figura 6.4). Nesta aba, existem duas subseções: uma para definir o mecanismo de troca de mensagens utilizado e a outra para o protocolo de sincronismo dos processos do programa de simulação.

Em relação ao protocolo para a troca de mensagens, o ambiente utiliza duas bibliotecas: MPI e PVM. Para o protocolo de sincronização, por padrão, o ambiente utiliza dois protocolos: *Rollback* Solidário e o *Time Warp*.

Seguindo a especificação proposta por Cruz (2009) em seu *framework*, é possível adicionar novos protocolos.

6.1.4 Aba Projetos

A aba projetos (figura 6.5) permite a visualização e execução de tarefas, como, por exemplo, adição e remoção de ítens que compõem o projeto de simulação. Ela é organizada através de uma árvore hierárquica, sendo o Projeto o grau mais alto de hierarquia. Abaixo de Projeto, há mais duas estruturas denominadas **Arquitetura** e **Modelos**, que abrigam os arquivos referentes ao projeto de simulação. Esta organização permite que sejam carregados e trabalhados mais de um projeto de simulação ao mesmo tempo.

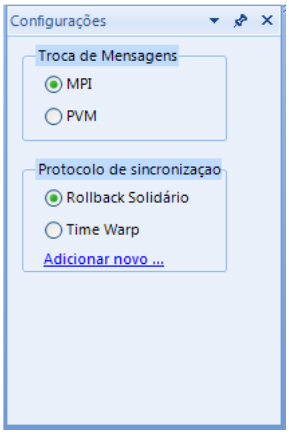

Figura 6.4: Configuração dos protocolos de troca de mensagens e sincronismo dos processos

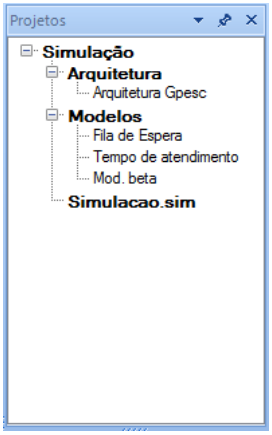

Figura 6.5: Organização Hierárquica dos *Projetos*

6.1.5 Aba Propriedades

A aba propriedades, apresentada na (figura 6.6), permite que o usuário visualize e modifique os atributos dos objetos dispostos no *sandBox*. Quando um objeto contido no *sandBox* é selecionado, automaticamente seus atributos são carregados na aba propriedades, para que que o usuário possa modificar os atributos desejados.

As abas laterais, presentes na ferramenta, podem ser facilmente retraídas para a margem da tela, através do ícone representado por um alfinete presente no cabeçalho de cada uma das abas, dessa maneira, deixando um área maior de visualização para o *sandBox*.

6.2 Funcionalidades e Operações

Esta seção descreve os aspectos funcionais da ferramenta. A criação de uma simulação, usando este ambiente, recebe o nome de **Projeto de Simulação**. Assim, pode-se definir

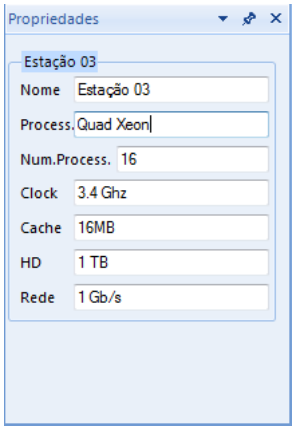

Figura 6.6: Propriedades de um objeto

o projeto de simulação como uma composição dos arquivos essenciais para a execução da simulação. Por padrão, foi definido que todo projeto deve conter os seguintes arquivos:

Arquitetura - arquivo com informações necessárias à definição e configuração do ambiente distribuído em que irá ocorrer a execução da simulação. Durante a simulação, este arquivo também é utilizado para o processo de monitoramento do ambiente, permitindo assim que o usuário visualize as informações referentes ao nó ou recurso de *hardware* selecionado.

Modelo - é o arquivo de maior importância, pois é por ele que o usuário consegue representar o sistema a ser simulado. Um único projeto de simulação pode ter mais de um modelo representando um mesmo sistema, porém com linguagens diferentes ou até mesmo diferentes sistemas.

Simulacao - gerado automaticamente após a escolha do modelo que será simulado. Contém a representação gráfica do modelo escolhido, fornecendo informações para recompor os componentes da interface, o que permitirá o acompanhamento e a visualização dos resultados da simulação.

6.2.1 Criação de um Projeto de Simulação

O assistente de criação (figura 6.7) auxilia o usuário a criar um projeto de simulação a partir de uma interface intuitiva. O usuário pode configurar um projeto em um ambiente distribuído rapidamente. Algumas configurações, como o protocolo de troca de mensagens e o protocolo de sincronismo, podem ser realizadas através deste assistente, como pode ser observado na figura 6.8.

Após a execução do assistente de criação, é criado um arcabouço do projeto na aba de projetos, contendo a estrutura básica para o desenvolvimento do trabalho. Com a criação

do arcabouço, o próximo passo tomado pelo usuário é a configuração da arquitetura e a criação dos modelos.

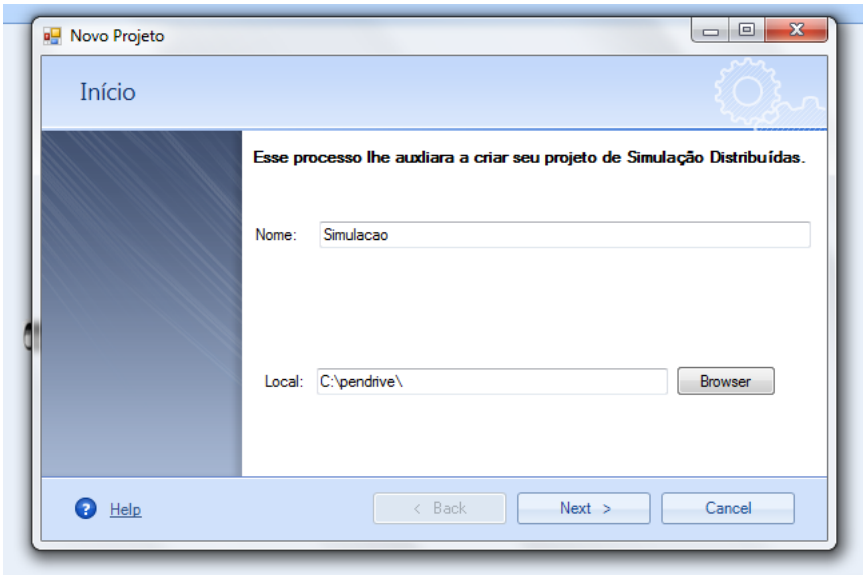

Figura 6.7: Assistente para criação de um Projeto de Simulação Distribuída

Os assistentes de criação de arquivos auxiliam o usuário no momento da criação de novos arquivos do tipo **Modelo** ou **Arquitetura**. A interface do assistente permite que sejam realizadas configurações referentes ao arquivo, por exemplo, em se tratando de um novo modelo, o assistente pergunta qual o tipo de linguagem simbólica que o usuário irá utilizar para a criação do modelo, a escolha do protocolo de sincronismo e da biblioteca de troca de mensagens são também escolhidas através de um assistente, como demonstrado na figura 6.8.

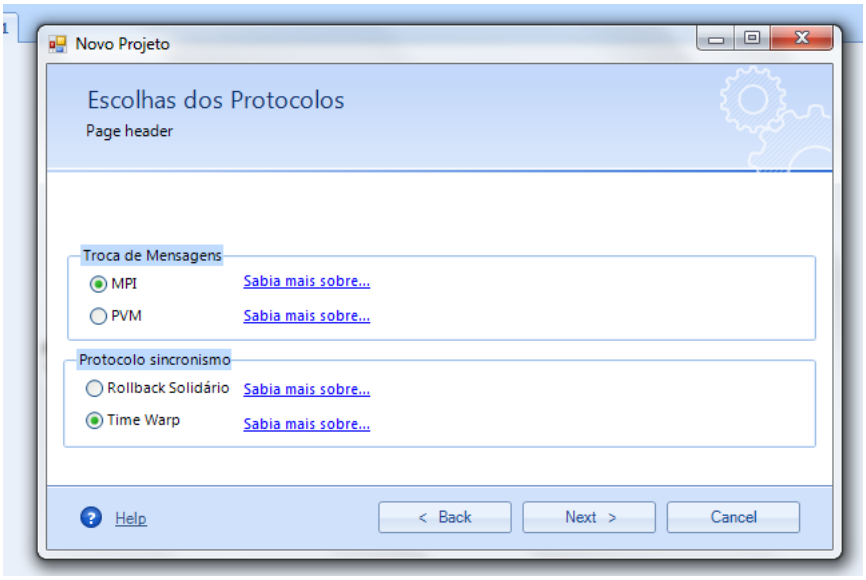

Figura 6.8: Assistente para criação de um projeto de simulação distribuída - configuração dos protocolos

6.2.2 Construção do Modelo

A construção do modelo é uma importante tarefa no processo de simulação. A ferramenta oferece apoio aos usuários durante a modelagem. O uso de ícones gráficos, que representam os elementos básicos da linguagem simbólica escolhida, permite a escolha destes elementos através de um *clique* do *mouse* e posicionamento sobre o *sandBox*.

Com um componente posicionado sobre o *sandBox*, o usuário deve configurar seus atributos e criar as conexões entre os componentes com os quais ele se relaciona. A criação dos relacionamentos entre os componentes é feita da seguinte maneira: o usuário deve selecionar um componente e *clicar* sobre o outro componente com o qual ele se relaciona, criando assim automaticamente uma ligação entre os dois componentes, o usuário deve prosseguir da mesma maneira para todos os componentes que apresentam relações entre si, criando assim o modelo desejado.

6.3 Execução da Simulação Distribuída

Após a criação do modelo e a configuração do ambiente, o projeto está apto a ser simulado em um *hardware* paralelo ou distribuído. A execução da simulação ocorre de forma transparente ao usuário, que não precisa implementar nenhuma linha de código para este fim. Outro fator importante, ainda no âmbito distribuído fornecido pela ferramenta, é o tratamento e detecção dos problemas inerentes à computação distribuída. Essa transparência permite que usuários em vários níveis de conhecimento sobre computação paralela se beneficiem do desempenho alcançado por um programa de simulação distribuído.

6.3.1 Um exemplo de simulação

Para entender como será realizada a execução paralela do programa de simulação, será necessário verificar a maneira como o *framework* (CRUZ, 2009) permite realizar esta tarefa. A partir de um arquivo XML com a estrutura do modelo, os processos lógicos são inicializados em cada nó da máquina paralela. Estes processos executam um algoritmo semelhante ao representado pelo fluxograma da figura 6.9.

Na implementação da ferramenta, o algoritmo executado pelos processos lógicos é semelhante, independente do modelo a ser simulado. Isso proporciona maior flexibilidade, favorecendo a adaptação do programa e a implementação das técnicas de mapeamento dinâmico dos processos. A diferença de comportamento reside nos dados do modelo que serão interpretados por cada processo do programa distribuído.

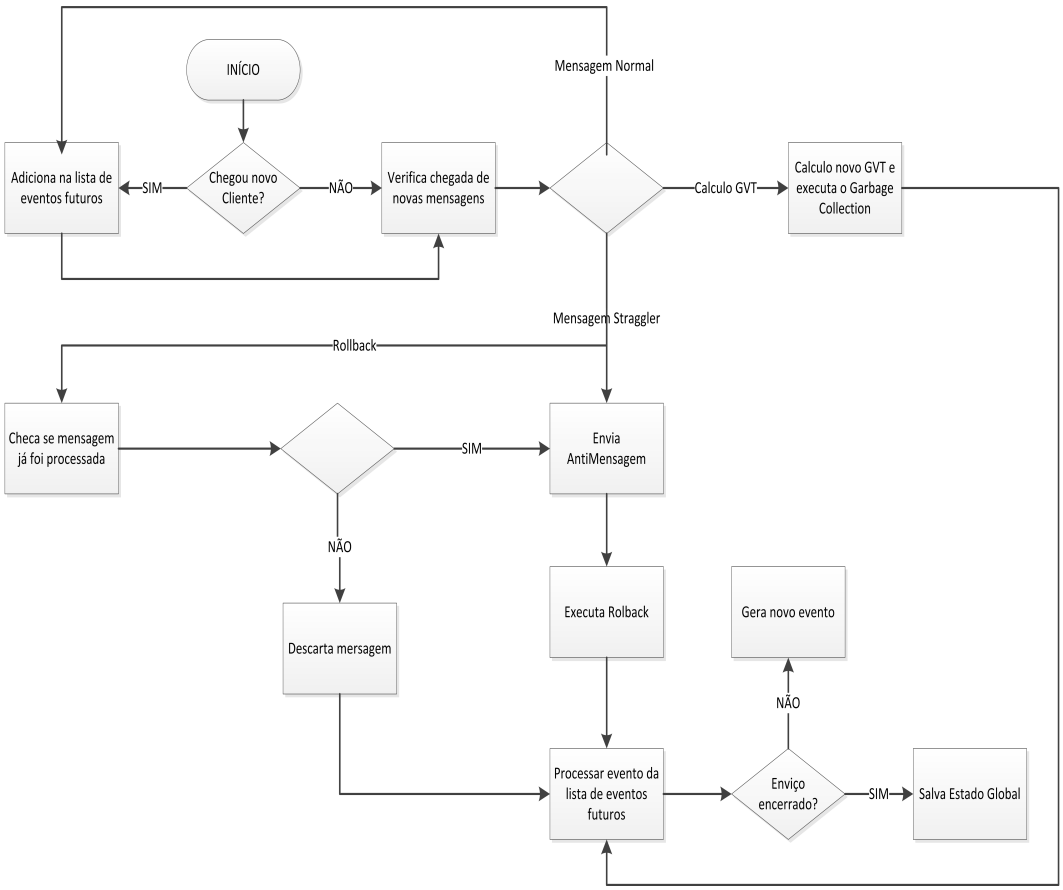

Figura 6.9: Diagrama de atividades de uma simulação

6.4 Considerações Finais

A elaboração do propotótipo funcional da interface apresentada neste capítulo, pôde auxiliar, na definição do arranjo visual da ferramenta.

A utilização da linguagem C# ofereceu recursos para a criação dos componentes gráficos, permitindo uma rápida prototipagem da interface escolhida.

O protótipo criado irá passar por testes de usabilidade, entre os pesquisadores do GPESC, a fim de avaliar os recursos oferecidos por ele. Após a conclusão desses testes, será realizada a transcrição final para a linguagem Java. A similaridade entre as duas linguagens permite que o protótipo possa ganhar sua verão final em Java sem maiores dificuldades. Destaca-se que já existe uma versão implementada em Java, utilizada para gerar os arquivos XML's a partir dos modelos e testar os protocolos do *framework*, mas que não contém todas as características e funcionalidades apresentadas neste capítulo.

Capítulo 7

Conclusões

As fases que compreendem o desenvolvimento de uma simulação podem trazer dificuldades para os usuários, por causa de alguns fatores, tais como, a falta de experiência do utilizador, ou até mesmo complicações devidas a complexidade e tamanho dos sistemas a serem modelados. No âmbito distribuído, essas dificuldades podem ser ainda maiores, pois as soluções disponíveis para tal propósito podem necessitar, por parte do usuário, conhecimento sobre computação distribuída.

O processo inicial da simulação é a modelagem do sistema, onde o usuário descreve como é o comportamento do sistema real que ele deseja simular. Para isso, deve fazer uso de alguma linguagem formal que permita essa especificação e consequente codificação em uma linguagem de programação. Porém, esse método pode ser um pouco inconveniente para o usuário, devido à necessidade de conhecimentos específicos de lógica de programção e, em particular, de uma linguagem de programação convencional, como C ou Java, por exemplo. Outra forma, é através de ferramentas que ofereçam recursos gráficos para modelagem, dessa forma, possibilitando que a tarefa de desenvolvimento do modelo seja executada de maneira mais fácil e ágil.

A simulação de grandes modelos, ou modelos complexos, exige a manipulção de grande quantidade de dados, fato que pode tornar a execução da simulação bastante demorada em um ambiente sequencial. Com utilização da computação distribuída, é possível obter um melhor desempenho durante a execução da simulação.

A tradução de um algoritmo sequencial para uma versão distribuída, não é uma tarefa trivial, pois a ocorrência de problemas já conhecidos no meio, como erros de causalidade, na qual o sistema não consegue garantir que os eventos de um mesmo processo sejam executados na sua ordem cronológica, podem comprometer a execução do programa. A principal causa desse problema, deve-se a falta de um relógio global de referência.

A utilização dos protocolos de sincronização, para os programas de simulação distri-

buída, tem como função evitar ou corrigir as ocorrências dos erros de causalidades que, eventualmente, ocontecem neste tipo de aplicação.

Além do problema de sincronização entre os processos, a simulação, em um ambiente distribuído, pode sofrer com o desbalancemanto da carga entre os processadores. Isso significa que há uma sobrecarga de processamento em um ou mais elementos de processamento, enquanto que existem outros trabalhando com pouca carga. Fatores como o avanço não homogêneo dos relógios lógicos dos processos, influência de outras aplicações e características internas da simulação podem ocasionar o desbalanceamento de carga em um programa de simulação distribuída, afetando o desempenho da simulação.

O monitoramento dos processos permite a coleta de informações que, ao serem analisadas, possibilitam a detecção do desbalanceamento de carga, permitindo a utilização de métodos para a migração dos processos, a fim de balancear o sistema dinamicamente.

A intrusividade causada pelo processo de monitoramento pode interferir no comportamento do sistema, seja na diminuição do desempenho, ou até mesmo na mudança da ordem global dos eventos, por isso deve-se adotar um mecanismo de monitoramento menos intrusivo possível, a fim de minimizar a interferência do processo monitor no desempenho da simulação.

7.1 Principais Contribuições deste Trabalho

A ferramenta apresentada nesta dissertação foi especificada para oferecer um ambiente gráfico unificado que auxilia o desenvolvimento de simulações em um ambiente distribuído, tratando os problemas descritos anteriormente. A ferramenta em questão provê o auxílio necessário para que seja possível a criação de um modelo, execução do mesmo em um ambiente distribuído e, também, recursos para o tratamento de erros de causalidade, monitoramento e balanceamento da carga dos processadores.

Com os recursos oferecidos na ferramenta apresentada, espera-se assim, fornecer aos seus usuários uma maior facilidade durante a criação de modelos e também durante as simulações dos mesmos. Outro fator importante esperado com a utilização da ferramenta é o ganho de desempenho durante a execução de grandes simulações através da computação paralela juntamente com o balanceamento de carga.

A escolha de uma arquitetura modularizada e orientanda a objetos permitiu estender o *framework* proposto por Cruz (2009), mantendo a proposta arquitetural já desenvolvida.

Finalmente, para que seja alcançada uma versão estável, será necessária a integração deste trabalho com outras pesquisas que estão em fase final de desenvolvimento por outros pesquisadores do grupo GPESC da UNIFEI.

7.2 Dificuldades Encontradas

Para a concretização deste trabalho, foram necessários unir diversos outros trabalhos que já se encontram finalizados e também em fase de desenvolvimento pelos integrantes do GPESC. Por este motivo, a conclusão total desta ferramenta fica a cargo das conclusões dos demais trabalhos necessários para a conclusão da ferramenta com todas as suas funcionalidades descritas nesta dissertação.

7.3 Trabalhos Futuros

Com o intuito de aprimorar as funcionalidades e qualidades apresentados neste trabalho, pode-se inumerar algumas sugestões para futuros trabalhos, como por exemplo:

- Adicionar as demais funcionalidades que se encontram em desenvolvimento pelo grupo, tais como: os protocolos de sincronismo e os algortimos de balanceamento de carga.
- Adaptar os algoritmos de migração de processos, elaborados por Junqueira (2012), permitindo, inclusive, que o usuário escolha qual processo ele deseja migrar e para onde ele irá.
- Desenvolver um módulo de depuração, possibilitando, ao usuário, o acompanhamento passo a passo da simulação.
- Implementar outros protocolos de sincronização.
- Desenvolver um mecanismo que permita a troca dinâmica dos protocolos de sincronização.
- Desenvolver mecanismos de animação gráfica da modelagem.
- Desenvolvimento de uma versão *web*, permitindo que os usuários simulem seus modelos a partir de qualquer dispositivo conectado à internet.
- Estudo comparativo sobre o desempenho da ferramenta proposta com as demais existentes no mercado.

Referências Bibliográficas

ADVANCED-PLANNING. *Simulation*. 2012. Http://www.advanced-planning.eu/advancedplanninge-375.htm.

BABAOGLU, O.; MARZULLO, K. Consistent global states of distributed systems: Fundamental concepts and mechanisms. In: MULLENDER, S. (Ed.). *Distributed Systems*. University of Bologna: Addison-Wesley, 1993. p. 55–96.

BABULAK, E.; WANG, M. Discrete event simulation. *Discrete Event Simulations*, p. 1–9, August 2010.

BELGE. *Simulando Com ProModel Curso Básico*. 2010. Http://yatex.files.wordpress.com/2010/09/apostila-pm.pdf.

BELL, P. C.; OKEEFE, R. M. Visual interactive simulation history, recent developments,and major issues. *Simulation*, p. 109–116, 1987.

BRANCO, K. R. L. J. C. *Índices de carga e desempenho em ambientes paralelos/distribuídos - modelagem e métricas*. Tese (Doutorado) — Universidade de São Paulo, São Carlos-SP, 2004.

BRUSCHI, S. M. *ASDA - um ambiente de simulação distribuída automático*. Tese (Doutorado) — Universidade de São Paulo, São Carlos-SP, 2003.

BRYANT, R. E. *Simulation of Packet Communication Architecture Computer Systems*. Massachussets, 1977. v. 7(3), 404–425 p. MIT-LCS-TR-188.

CAROTHERS, C. D.; FUJIMOTO, R. M. Efficient execution of time warp programs on heterogeneous now platforms. *IEEE Transactions on Parallel and Distributed Systems*, v. 11, n. 3, 2000.

CASTILHO, M. R. *O uso da simulação computacional como ferramenta de auxílio à tomada de decisão : aplicação em empresa de papelão ondulado*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul., 2004.

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

CHWIF, L.; MEDINA, A. C. *INTRODUÇÃO AO SOFTWARE DE SIMULAÇÃO SIMUL8*. 2006. `http://www.livrosimulacao.eng.br/download/Sobrapo_Simul8_.pdf`.

CHWIF, L.; MEDINA, A. C. *Modelagem e Simulação de Eventos Discretos*. [S.l.]: Ed. dos Autores 2006, 2007.

COSTA, L. C. Teoria das filas. *Universidade Federal do Maranhão.*, p. 1–61, 2006.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. *Distributed Systems - Concepts and Design*. 3. ed. [S.l.]: Addison-Wesley Publishing Company Inc., 2001. 772 p.

CRUZ, L. B. da. *Projeto de Um Framework para o Desenvolvimento de Aplicações de Simulação Distribuída*. Dissertação (Mestrado) — UNIFEI, Itajubá-MG, 2009.

DAVIS, D. A.; PEGDEN, C. D. Introduction to siman. In: *Proceedings of the 20th conference on Winter simulation*. New York, NY, USA: ACM, 1988. p. 61–70.

DIACONESCU, R.; CONRADI, R. A data parallel programming model based on distributed objects. In: *Cluster Computing, 2002. Proceedings. 2002 IEEE International Conference on*. [S.l.: s.n.], 2002. p. 455–460.

DONGARRA, J. et al. (Ed.). *Sourcebook of parallel computing*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

DYNAMICS, I. E. *Tutorial ED 7*. 2006. `http://www.cs.uu.nl/docs/vakken/scm/Tutorial%20ED.pdf`.

FLEISCHMANN, J.; WILSEY, P. A. Comparative analysis of periodic state saving techniques in time warp simulators. *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*, v. 25, n. 1, p. 50–58, July 1995.

FUJIMOTO, R. M. Distributed simulation systems. *Proceedings of the 2003 Winter Simulation Conference*, p. 124–134, 2003.

FUJIMOTO, R. M.; MALIK, A. W.; PARK, A. J. Parallel and distributed simulation in the cloud. *Science*, Wiley Inter Science, v. 3, n. 3, p. 1–10, 2010.

GAMMA, E. et al. *Padrões de Projeto - Soluções Reutilizáveis de Software Orientado a Objetos*. Porto Alegre-RS: Bookman, 2007.

GARCíA, A.; GARCíA, I. A simulation-based flexible platform for the design and evaluation of rail service infrastructures. *Simulation Modelling Practice and Theory*, v. 27, p. 31 – 46, 2012. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1569190X12000676>.

GEIST, A. et al. *PVM: Parallel Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing*. [S.l.]: The MIT Press, 1994.

HAMMOND, S. D. et al. Warpp - a toolkit for simulating high-performance parallel scientific codes. In: *INTERNATIONAL CONFERENCE ON SIMULATION TOOLS AND TECHNIQUES (SIMUTOOLS 2009)*. [S.l.]: ICST, 2009.

HAREL, D. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, p. 231 – 274, 1987.

INCONTROL. *Simulation Software / TUTORIAL*. 2009. Http://www.incontrolsim.com/.

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

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

LIMITED, A. S. *Rapiscan Systems Dashboard software use AutoMod to perform real time customisation of models*. 2012. Http://www.autologic-systems.co.uk/index.php/news/37/24/Rapiscan-use-AutoMod-modelling-software.

LOBãO, E. de C.; PORTO, A. J. V. Evolução das técnicas de simulação. scielo, p. 13 – 21, 06 1999.

MARKOVITCH, N. A.; PROFOZICH, D. M. Arena software tutorial. In: *Proceedings of the 28th conference on Winter simulation*. Washington, DC, USA: IEEE Computer Society, 1996. (WSC '96), p. 437–440. ISBN 0-7803-3383-7. Disponível em: <http://dx.doi.org/10.1145/256562.256708>.

MIYAGI, P. E. Introdução à simulação discreta. In: *Apostila do curso Modelagem e Controle de Sistemas Discretos*. [S.l.: s.n.], 2006.

MOREIRA, E. M. *Rollback Solidário: Um Novo Protocolo Otimista para Simulação Distribuída*. Tese (Doutorado) — Universidade de São Paulo, São Carlos-SP, 2005.

MOREIRA, E. M. et al. Um framework para o desenvolvimento de programas de simulação distribuída. In: *42o. Simpósio Brasileiro de Pesquisa Operacional*. Bento Gonçalves-RS: [s.n.], 2010.

MOREIRA, E. M.; SANTANA, R. H. C.; SANTANA, M. J. Using consistent global checkpoints to synchronize processes in distributed simulation. *Proceedings of the 9th The 9th IEEE International Symposium on Distributed Simulation and Real Time Applications*, p. 43–50, 2005.

MURATA, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, p. 541–580, 1989.

NAKASONE, A. et al. Openenergysim: a 3d internet based experimental framework for integrating traffic simulation and multi-user immersive driving. In: *SimuTools*. [S.l.: s.n.], 2011.

NUNEZ, A. et al. Simcan: a simulator framework for computer architectures and storage networks. In: . ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008. p. 73:1–73:8.

ONISHI, A. E. *Técnicas de Reuso de Software aplicados na elaboração de Arquiteturas Corporativas*. 2006.

PAM, L. M.; MICHAEL, H. M. Witness simulation software a flexible suite of simulation tools. In: *Simulation Conference, 1997. Proceedings of the Winter*. [S.l.: s.n.], 1997.

PARK, A.; FUJIMOTO, R. A scalable framework for parallel discrete event simulations on desktop grids. In: *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2007. (GRID '07), p. 185–192.

PARK, A.; FUJIMOTO, R. M.; PERUMALLA, K. S. Conservative synchronization of large-scale network simulations. In: *Proceedings of the eighteenth workshop on Parallel and distributed simulation*. New York, NY, USA: ACM, 2004. (PADS '04), p. 153–161. ISBN 0-7695-2111-8.

PEGDEN, C. D. Simio: a new simulation system based on intelligent objects. In: *Proceedings of the 39th conference on Winter simulation: 40 years! The best is yet to come*. Piscataway, NJ, USA: IEEE Press, 2007. p. 2293–2300.

RICE, S. V. et al. Object-oriented simscript. In: *Proceedings of the 37th annual symposium on Simulation*. Washington, DC, USA: IEEE Computer Society, 2004. p. 178–.

ROHRER, M. W.; MCGREGOR, I. W. Simulating reality using automod. In: *Simulation Conference, 2002. Proceedings of the Winter*. [S.l.: s.n.], 2002. p. 173 – 181 vol.1.

SAKURADA, N.; MIYAKE, D. I. Estudo comparativo de softwares de simulação de eventos discretos aplicados na modelagem de um exemplo de loja de serviços. In: *XXIII Encontro Nac. de Eng. de Produção, 2003*. [S.l.: s.n.], 2003. p. 173 – 181 vol.1.

SARJOUGHIAN, H. S.; ELAMVAZHUTHI, V. Cosmos: A visual environment for component-based modeling, experimental design, and simulation. *Proceedings of the International Conference on Simulation Tools and Techniques*, p. 43–50, 2009.

SCHNEIDER, C. S. *Utilização dos Aspectos Ergonômicos na Simulação de Sistemas de Produção*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul, 2004.

SIMIO. *Case Study: Simio Models Movement of Armed Forces Between Germany and Afghanistan*. 2012. Http://www.simio.com/case-studies/Konekta/.

SNIR, M. et al. *MPI: The Complete Reference*. [S.l.]: The MIT Press, 1996.

SOBEIH, J. C. H. A. A simulation framework for sensor networks in j-sim. In: . [S.l.]: Department of Computer Science University of Illinois at Urbana-Champaign, 2008.

SONODA, E.; TRAVIESO, G. The oops framework: high level ions for the development of parallel scientific applications. In: *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. New York, NY, USA: ACM, 2006. (OOPSLA '06), p. 659–660.

SOUZA, M. A. de et al. Sistema de monitoração para o escalonamento de processos: Estrutura e métricas de desempenho. *Revista Unieuro de Tecnologia da Informação*, v. 1, n. 1, p. 5–13, 2008.

SPOLON, R. *Um Método para Avaliação de Desempenho de Procotolos de Sincronização Otimistas para Simulação Distribuída.* Tese (Doutorado) — Universidade de São Paulo, São Carlo-SP, 2001.

STANDISH, R. K.; MADINA, D. Classdescmp: Easy mpi programming in c++. *CoRR*, cs.DC/0401027, 2004.

SUZUKI, J. inet: an extensible framework for simulating immune network. In: . [S.l.]: ieee, 2000.

TESSER, R. K. *Monitoramento on-line em Sistemas Distribuídos: Mecanismo Hierárquico para Coleta de Dados*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul - UFRGS, 2011.

WEST, J.; MULLARNEY, A. *ModSim: A language for distributed simulation*. Feb 03 1988. 159 p. Disponível em: <http://search.proquest.com/docview/24949313?accountid=26619>.

WISSINK, A. M.; HYSOM, D.; HORNUNG, R. D. Enhancing scalability of parallel structured amr calculations. In: *Proceedings of the 17th annual international conference on Supercomputing*. New York, NY, USA: ACM, 2003. (ICS '03), p. 336–347.

WITNESS. *WITNESS 12 - Making it easier to provide answers for business critical problems*. 2012. `http://www.lanner.com/en/media/witness/witness12.cfm`.

YAU, V. Automating parallel simulation using parallel time streams. *ACM Trans. Model. Comput. Simul.*, v. 9, p. 171–201, 1999.