Utilização de queuing statecharts em um ambiente de simulação automático

Thais Souza Aiza

Orientador: *Prof. Dra. Regina Helena Carlucci Santana*

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências da Computação e Matemática Computacional.

“VERSÃO REVISADA APÓS A DEFESA”

Data da Defesa: 11/04/2005

Visto do Orientador: *Regina Carlucci Santana*

USP – São Carlos
Abril/2005

Sumário

Lista de Figuras	vi
Lista de Tabelas	vii
1 Introdução	3
2 Avaliação de Desempenho	7
2.1 Considerações Iniciais	7
2.2 Técnicas de Avaliação de Desempenho	8
2.2.1 Técnicas de Aferição	8
2.2.2 Técnicas de Modelagem	11
2.2.3 Técnicas para Especificação de Modelos	12
2.2.4 Soluções para o Modelo	13
2.3 Considerações Finais	15
3 Representação dos Modelos	17
3.1 Considerações Iniciais	17
3.2 Redes de Filas	17
3.2.1 Notação para Sistemas de Filas	18
3.2.2 Classificação das Redes de Filas	20
3.2.3 Parâmetros de Desempenho e Restrições em Redes de Filas	21
3.3 Redes de Petri	22
3.3.1 Redes de Petri Marcadas	24
3.3.2 Redes de Petri Elementares	25
3.3.3 Redes de Petri Coloridas	27
3.3.4 Redes de Petri Estocásticas	27
3.3.5 Redes de Petri Estocásticas Generalizadas	28
3.3.6 Vantagens e Desvantagens de Redes de Petri	29
3.4 <i>Statecharts</i>	30
3.4.1 Definição e Notação Básica	31
3.4.2 Estado <i>Default</i>	31
3.4.3 Entrada por Condição	32
3.4.4 <i>Delays e Timeouts</i>	32
3.4.5 Estados Parametrizados	33
3.4.6 Semântica dos <i>Statecharts</i>	33
3.4.7 Vantagens e desvantagens dos <i>Statecharts</i>	36
3.5 <i>Statecharts</i> Estocásticos	36
3.5.1 Redefinição de Passos e Configurações para os <i>Statecharts</i> Estocásticos	37
3.5.2 Templates e Eventos-Padrão para Sistemas de Filas	40

3.6	Queuing Statecharts	42
3.6.1	Definição de <i>Queuing Statecharts</i>	43
3.6.2	Templates e Eventos-Padrão para Queuing Statecharts	44
3.7	Considerações Finais	45
4	ASDA	49
4.1	Considerações Iniciais	49
4.2	Simulação Distribuída	50
4.3	ASDA	51
4.3.1	Módulo de Interface com o Usuário	53
4.3.2	Módulo Avaliador	58
4.3.3	Módulo Gerador	59
4.4	Considerações Finais	59
5	Implementação do Módulo de Interface com o Usuário	61
5.1	Considerações Iniciais	61
5.2	A Escolha da Técnica de Especificação Gráfica	62
5.3	A Escolha da Ferramenta para a Implementação	62
5.4	Base de Dados do ASDA	64
5.5	Definição das Estrutura de Dados do Modelo	66
5.5.1	Graph	68
5.5.2	Node	68
5.5.3	Arc	70
5.6	A Interface Gráfica do ASDA	70
5.6.1	Editor Gráfico	72
5.6.2	Parametrização do Modelo de Simulação	76
5.6.3	Consistência do Modelo - Módulo Avaliador Nível 1	78
5.7	Considerações Finais	80
6	Implementação do Gerador do Programa de Simulação	83
6.1	Considerações Iniciais	83
6.2	Gerador	83
6.3	Execução do Gerador e Geração do Código	87
6.3.1	Fase de Definições	88
6.3.2	Fase de Preparação do Modelo	90
6.3.3	Fase de Implementação	90
6.4	Exemplos de Programas Produzidos pelo Gerador	92
6.4.1	Exemplo 1	92
6.4.2	Exemplo 2	96
6.5	Considerações Finais	100
7	Conclusão	101
7.1	Conclusões Gerais	101
7.2	Contribuições do Trabalho	102
7.3	Sugestões para Trabalhos Futuros	103
	Referências Bibliográficas	105

Lista de Figuras

2.1	Etapas do Processo de Modelagem	12
2.2	Uso Preferencial das Técnicas de Modelagem	16
3.1	Centro de Serviço	18
3.2	Redes Abertas e Fechadas	20
3.3	Tipos de Absorção de Centros de Serviço	21
3.4	Servidor de Arquivos Representado em Rede de Filas	21
3.5	Descrição dos Elementos Básicos de Redes de Petri	23
3.6	Exemplo de Representação Gráfica de uma Rede de Petri	23
3.7	Disparo de Transição	25
3.8	Sequenciamento	25
3.9	Distribuição	26
3.10	Junção	26
3.11	Escolha Não-Determinística	27
3.12	Rede de Petri Colorida de um Cliente TCP/IP	28
3.13	Representação Gráfica de Transições	29
3.14	Servidor de Arquivos em GSPN	30
3.15	Estado <i>Default</i>	31
3.16	Entrada por Condição	32
3.17	Estado com <i>Delay</i> de 2 Segundos	32
3.18	Estados Parametrizados Processor/Disc	33
3.19	Servidor de Arquivos na Especificação <i>Statecharts</i>	34
3.20	Estados com Retardo e Imediatos (u.t.: unidades de tempo)	37
3.21	Escolha por Probabilidade	37
3.22	Servidor de Arquivos e seus Estados com Retardos Associados	39
3.23	<i>Templates</i> para Fontes Geradoras de Clientes	40
3.24	<i>Template</i> para Fila de Servidores	41
3.25	<i>Template</i> para Servidor	41
3.26	<i>Template</i> para Sorvedouro	42
3.27	<i>Template</i> para Fonte (<i>Source</i>)	44
3.28	<i>Template</i> para Fila (<i>Queue</i>)	45
3.29	<i>Template</i> para Servidor (<i>Server</i>)	45
3.30	<i>Template</i> para Centro de Serviço (<i>Service Center</i>)	45
3.31	<i>Template</i> para Sorvedouro (<i>Destination</i>)	45
4.1	Diagrama Modular do ASDA	53
4.2	Estrutura de Arquivos no Módulo Gerador	59

5.1	Tabela de Usuários do ASDA	65
5.2	Diagrama de Classes do Modelo	66
5.3	Exemplo de Representação Gráfica	67
5.4	Estrutura de Dados do Modelo	67
5.5	Janela de Autenticação do Usuário no ASDA	71
5.6	Janela de Saudação para o SuperUsuário	72
5.7	Janela de Saudação para o Usuário de Simulação	72
5.8	Cadastro de Usuário	72
5.9	Janela de Edição Gráfica	74
5.10	Barra de Ferramentas Horizontal	74
5.11	Barra de Ferramentas Vertical	75
5.12	Módulo Avaliador Nível 1 - Verificação da Consistência da Especificação Gráfica	79
5.13	Módulo Avaliador Nível 1A - Verificação da Consistência da Especificação Gráfica	80
5.14	Módulo Avaliador Nível 1B - Verificação da Consistência da Parametrização do Modelo	80
5.15	Armazenamento do Modelo	80
6.1	Gabarito	84
6.2	Trecho de Código da Leitura do Gabarito pelo Gerador	85
6.3	Estrutura de Arquivos no Módulo Gerador	86
6.4	Execução do Gerador do Programa de Simulação	87
6.5	Modelo para Exemplificação do Código Gerado	88
6.6	Exemplo 1	93
6.7	Exemplo 1	96
6.8	Relatório Final Exemplo 2	96
6.9	Relatório Final Exemplo 2	100

Lista de Tabelas

3.1	Eventos-Padrão e suas Semânticas	42
3.2	Nomes de Estados-Padrão e suas Semânticas	43
3.3	Eventos-Padrão de <i>Queuing Statecharts</i> e suas Semânticas	46
3.4	Características das Técnicas de Especificação	47
5.1	Comparação do MySQL com outros produtos de banco de dados relacional	64
5.2	Estrutura da tabela <i>usuario</i>	65

RESUMO

Este trabalho apresenta a utilização de *Queuing Statecharts* para a especificação de modelos de sistemas computacionais em um sistema de simulação automático. Durante o trabalho de mestrado, foi implementado o Módulo de Interface Gráfica com o Usuário e o Módulo Avaliador Nível 1 do ASDA (Ambiente de Simulação Automática Distribuída), e o desenvolvido um gerador de programas de simulação para a extensão funcional SMPL (*Simulation Programming Language*). Foram definidas as estruturas de dados que permitem o armazenamento, a alteração e recuperação do modelo. O módulo de Interface com o Usuário implementado permite que o usuário defina o modelo, a parametrização e o gerenciamento das informações dos usuários do ambiente que atua em conjunto com o Módulo Avaliador Nível 1 consiste na definição das regras para a verificação da consistência do modelo e a validação dos parâmetros. O gerador de programas de simulação desenvolvido possibilita que o usuário gere automaticamente seus programas de simulação a partir da especificação gráfica e dos parâmetros do modelo definidos pelo usuário de simulação. As regras para a verificação da consistência do modelo e da validação dos parâmetros foram definidas durante o desenvolvimento do Módulo Avaliador Nível 1. O gerador para programas de simulação desenvolvido neste trabalho permite ao usuário utilizar o modelo armazenado na estrutura de dados e gerar automaticamente o seu programa de simulação. São apresentados exemplos e resultados de programas de simulação gerador.

ABSTRACT

This master thesis presents a work consisting in including the Queuing Statecharts approach for specifying computational system models for performance evaluation. During this work, the User Graphic Interface Module and the Evaluation Level 1 Module for ASDA (Automatic Distributed Simulation Environment) were implemented and a generator of simulation programs for the functional extension SMPL (Simulation Programming Language) was developed. The data structure which handles the model storage, modification and recovering was also defined. The User Graphic Interface Module allows the user to define the model and parameters and to manage the information about the users in the environment, acting together the Evaluation Level 1 Module for make it possible to user to define the model and parameters without inconsistencies. The development and implementation of Evaluation Level 1 Module consisted in defining the rules for verifying the model consistence and validating the parameters. The generator of simulation programs allows the user to automatically generate simulation programs from graphic specification and the model parameters defined by the simulation user. The rules for verifying the model consistence and validating the parameters were defined during the development of Evaluation Level 1 Module. The generator of simulation programs developed in this work allows the user to utilize the model stored in the data structures and to generate automatically simulation programs. Examples and results of generated simulation programs are presented.

Agradecimentos

A Deus por mais essa realização, por todas as oportunidades que tive até hoje e pela saúde.

Aos meus queridos pais, Maria e Ademir Aiza que com tanto amor me educaram e me ensinam fundamentos que levarei comigo para sempre. A eles agradeço pelos apoios morais, emocionais e financeiros incondicionais.

À minha orientadora, Regina Helena Carlucci Santana, por ter confiado na minha capacidade de conduzir esse projeto; pela constante motivação e perseverança com que encarou os desafios deste trabalho e pelo prazer de compartilhar da sua amizade e do carinho que demonstrou ter por mim.

Aos meus irmãos Laice e Agnaldo Aiza pelo carinho, amor, apoio e compreensão, e principalmente por entenderem os momentos de minha ausência.

Ao Alex Carvalho pelo amor e compreensão que demonstrou durante os meus momentos de ausência e nervosismo, principalmente pela sua compreensão e incentivo para que eu pudesse continuar em frente.

A Sarita Bruschi e ao Professor Marcos Santana, pela amizade, pelas dicas, conselhos e os trabalhos realizados em parceria.

A minha grande amiga Patricia Oliveira pela amizade fiel, conselhos valiosos e pelos grandes momentos de diversão e alegria.

As amigas Eliane Tatsumi e Valéria Quadros pela amizade e pelo apoio, sem os quais não seria possível a finalização deste trabalho.

As amigas Kalinka Branco e Célia Ogawa e ao amigo Hermes Pimenta Júnior pelos momentos compartilhados dentro e fora da USP.

A Beth (secretária de pós-graduação) pelas dicas e Maria (biblioteca) pela simpatia e compreensão com que sempre me atenderam.

Aos meus professores da Universidade Federal de Mato Grosso pelo conhecimento transmitido durante os nove semestres de duração do curso de bacharel em Ciência da Com-

putação, em especial ao Professor Alexandre Matos.

Ao CNPq pelo apoio financeiro.

Introdução

Com a evolução tecnológica experimentada pelos computadores nas últimas décadas, surgiram máquinas cada vez mais velozes e as exigências em relação ao desempenho, confiabilidade, interface com o usuário, velocidade de comunicação, tolerância a falhas, entre outros, têm sido cada vez maiores. O desempenho constitui um dos fatores cruciais de um sistema computacional, seja ele distribuído ou não.

A avaliação de desempenho de um sistema computacional não é uma tarefa fácil de ser conduzida. Quando se pretende avaliar um sistema complexo, o primeiro desafio está em se encontrar a técnica adequada para levar a cabo a avaliação. A técnica adotada não deve interferir no desempenho do sistema a ser avaliado e deve oferecer os dados necessários para a avaliação.

A maneira pela qual será avaliado o desempenho, depende diretamente das características do sistema envolvido. Em se tratando de sistemas computacionais, pode-se dividir a avaliação em duas abordagens distintas. A primeira consiste na obtenção de medidas no próprio sistema, e a partir delas o avaliador estabelece uma interpretação em relação ao desempenho. A segunda abordagem se baseia em criar uma abstração do sistema, representando-o através de um modelo, utilizado para estabelecer um prognóstico a respeito do desempenho do sistema.

Uma taxonomia para abordagens para avaliação de desempenho é proposta em (Santana *et al.*, 1997), onde seus autores sugerem uma divisão nas duas abordagens citadas: as técnicas de aferição e as técnicas de modelagem, sendo que um grande diferencial entre os dois grupos está em ter-se ou não o sistema implementado.

Para os casos em que o sistema já existe e, conseqüentemente, medidas podem ser realizadas no próprio sistema, as técnicas de aferição podem ser recomendadas. Estão nessa

classe, por exemplo, a utilização de *benchmarks*, os protótipos e a coleta de dados.

Para o caso em que se deseja avaliar um projeto, em que o sistema ainda não está disponível, as técnicas mencionadas no parágrafo anterior tornam-se inviáveis. Neste caso, a utilização de técnicas de modelagem pode auxiliar no entendimento do projeto, na sua validação e na avaliação do desempenho que será obtido (Swain, 1997). Uma outra situação em que técnicas de modelagem podem ser utilizadas com grandes vantagens é na avaliação de modificações em sistemas existentes. Esta técnica é capaz de responder de forma tanto quantitativa quanto qualitativa às perguntas feitas constantemente pelos projetistas ou administradores de sistemas. Como um exemplo desta situação pode-se citar a avaliação da influência de um processador duas vezes mais rápido no desempenho de uma máquina. A modelagem é uma técnica flexível e que pode ser utilizada em diferentes situações.

O processo de desenvolvimento de uma avaliação por modelagem envolve diversas etapas. Primeiramente é necessário especificar o modelo, abstraindo as características essenciais do sistema a ser avaliado. Esse modelo pode então ser resolvido através da solução analítica ou por simulação.

Para especificar o modelo do sistema a ser avaliado, pode-se utilizar diversas abordagens: redes de filas, redes de Petri, Statecharts, etc. As redes de filas são as mais utilizadas para especificar modelos, visando avaliação de desempenho de sistemas computacionais. No entanto, esta abordagem apresenta uma série de limitações, tornando difícil de serem representadas diversas características normalmente encontradas em sistemas computacionais. Francês, em (Francês, 2001) propôs novas abordagens para esta representação - os *Statecharts* Estocásticos e os *Queuing Statecharts*.

O objetivo deste trabalho de mestrado é a inclusão da abordagem *Queuing Statecharts* em um Ambiente de Simulação Distribuída Automático (ASDA) (Bruschi, 2002). Esta inclusão envolve: o desenvolvimento de uma interface gráfica que permita ao usuário definir o modelo a ser simulado em *Queuing Statecharts* armazenamento dos dados do modelo em um arquivo e, finalmente, a geração do programa de simulação.

Inicialmente foi efetuada uma avaliação da abordagem *Queuing Statecharts* que visou verificar a adequabilidade da abordagem em um ambiente de simulação automática onde deverá ser especificado o modelo a ser simulado.

Assim, objetivo do trabalho realizado também inclui a construção do Módulo de Interface com o Usuário do ASDA que é a parte do ambiente que interage com o usuário. Inclui ainda a construção de um Módulo Avaliador Nível 1 que contém as definições das regras para a verificação da consistência da especificação gráfica e da parametrização do modelo. O gerador de programas de simulação foi implementado utilizando-se a linguagem

de programação Java (Gosling & McGilton, 1996), a ferramenta Eclipse (Eclipse, 2001) e o sistema gerenciador de banco de dados MySQL (Suchring, 2002) e gera programas na extensão funcional SMPL (*Simulation Programming Language*) (Macdougall, 1987).

Esta dissertação apresenta a revisão bibliográfica utilizada e descreve o trabalho desenvolvido durante o programa de mestrado, estando organizada da seguinte forma: o Capítulo 2 aborda a avaliação de desempenho, apresentando as técnicas existentes. O Capítulo 3 traz as principais técnicas de representação dos modelos mais citadas na literatura especializada e duas novas abordagens para especificar os modelos, um dos objetivos do estudo deste trabalho de mestrado.

O Capítulo 4 apresenta o ASDA e descreve com maiores detalhes os módulos que estão envolvidos no trabalho de mestrado. O Capítulo 5 encontra-se uma descrição do trabalho feito para a implementação do Módulo de Interface Gráfica, bem com a justificativa da escolha da técnica de modelagem a ser incluída no ASDA, a linguagem de programação para a implementação e o sistema gerenciador de banco de dados e a definição da estrutura de dados para o armazenamento do modelo. A implementação do gerador de programa de simulação e os exemplos de modelos gerados são descritos no Capítulo 6. O Capítulo 7 traz as conclusões e contribuições deste trabalho de mestrado e sugestões para trabalhos futuros.

Avaliação de Desempenho

2.1 Considerações Iniciais

A evolução tecnológica, experimentada nas últimas décadas, tornou os sistemas cada vez mais complexos e, ao mesmo tempo, o usuário ficou mais exigente em relação aos sistemas que lhe propiciam algum serviço. Desta forma, torna-se cada vez mais importante a utilização de técnicas que permitam a avaliação do desempenho dos sistemas computacionais.

Desempenho em ciência da computação e engenharia, geralmente, refere-se a quão efetivamente (por exemplo, *throughput*, *delay*) ou eficientemente (por exemplo, utilização de recurso) um sistema entrega um serviço específico, presumindo que ele foi corretamente entregue (Meyer, 1995). O objetivo da avaliação de desempenho é prover os dados e percepções requeridas, para otimizar a execução da aplicação ou de sistemas computacionais.

A avaliação de desempenho em sistemas distribuídos pode ser aplicada na análise de sistemas existentes, na de sistemas ainda em desenvolvimento ou na seleção de um determinado sistema (Santana, 1990). Nos sistemas existentes, a análise de desempenho compreende a maximização da eficiência e da utilização dos recursos, processamento a um custo mínimo, diminuição do tempo de resposta, etc. Em sistemas inexistentes a avaliação de desempenho deve fazer previsões sobre o comportamento do novo sistema. Para a seleção de um novo sistema é feita uma comparação entre os desempenhos desses sistemas e a escolha é realizada de acordo com um parâmetro significativo no processo de avaliação de desempenho (Orlandi, 1995).

Dependendo do tipo de avaliação desejada e do tipo de sistema em consideração, é possível o envolvimento com ferramentas teóricas (modelagem analítica do sistema e solução das equações obtidas), simulação (desenvolvimento e implementação de um modelo de

simulação que represente com fidelidade o sistema real) ou ainda experimentação prática, para sistemas existentes (instrumentação para coleta de dados e posterior análise) (Santana *et al.*, 1994).

2.2 Técnicas de Avaliação de Desempenho

A escolha da técnica a ser utilizada na avaliação de desempenho é uma etapa importante. Dependendo do tipo de avaliação desejada e o tipo de sistema em consideração, as informações requeridas por um estudo, podem ser obtidas a partir do próprio sistema (técnicas de aferição - *benchmark*, coleta de dados e construção de protótipos) ou então por meio de um modelo representativo do sistema (modelagem analítica ou simulação) (Santana *et al.*, 1994).

As técnicas de aferição podem apresentar os resultados mais precisos, uma vez que, ela avalia o sistema por meio de observações diretamente no sistema real em funcionamento, mas exigem que o sistema a ser avaliado esteja disponível, aos menos, como um protótipo, caso contrário não é possível realizar as medidas diretamente (Jain, 1991). Essa técnica exige o uso de monitores para extrair dados para o estudo, e tem um custo geralmente elevado, quando comparado com as outras técnicas de avaliação. Como há a exigência física do sistema a ser avaliado por esse motivo, nem sempre pode ser aplicada.

As técnicas de modelagem não dependem diretamente da existência do sistema real. Elas baseiam-se em modelagens em níveis diferentes de abstração do sistema a ser avaliado, e portanto podem ser executadas antes que um protótipo esteja pronto. Mas por representarem abstrações da realidade, simplificações são inseridas nos modelos, e então a precisão da avaliação pode ser em parte prejudicada. Uma vez definido um modelo, este pode ser avaliado através de soluções analíticas ou por simulação. A solução analítica exige um domínio matemático, porém é uma técnica de baixo custo e rápida.

A simulação apresenta uma qualidade de resultados superior à técnica analítica, pois considera um modelo mais apurado do sistema, sendo possível especificar mais detalhes no modelo. Ela não interage diretamente com o sistema que está sendo avaliado e consome mais tempo de construção da simulação, que deve ser especificado em uma linguagem de simulação.

2.2.1 Técnicas de Aferição

As técnicas de aferição são utilizadas quando o sistema computacional já existe ou está em fase final de desenvolvimento. Com a experimentação do sistema pode-se deduzir o

seu desempenho (Jain, 1991).

Técnicas de aferição não são tarefas fáceis de serem conduzidas. Porém, fornecem resultados mais precisos que as técnicas de modelagem. Em situações em que o sistema não existe e é necessário a realização de testes prognósticos sobre o sistema em desenvolvimento, elas não podem ser empregadas.

Duas dificuldades são encontradas na experimentação do sistema. A primeira é como experimentar o sistema, de modo que a própria experimentação não influencie no comportamento do sistema e, portanto, não interfira na análise dos resultados. A segunda dificuldade é quando há necessidade de se verificar as diferentes alternativas de sistemas, pois implica em altos custos para alterações e testes, além da dificuldade de executá-la.

Coleta de Dados

Entre todas as técnicas de avaliação de desempenho, a coleta de dados é a técnica que oferece os resultados mais precisos. Essa técnica envolve uma monitoração do sistema, enquanto ele está sendo submetido a uma carga em particular. Esta monitoração pode ser feita usando-se software ou hardware específico e, para que as medições efetuadas sejam significativas, a carga a que o sistema é submetido deve ser cuidadosamente selecionada. A carga pode ser real ou sintética. Com carga real, a observação é feita utilizando um sistema em sua operação normal e não pode ser repetida nas mesmas condições. Já a carga sintética é gerada artificialmente, apresentando características similares àquelas da carga real, com a vantagem de poder ser aplicada repetidamente e de maneira controlada (Tatsumi, 2002).

Há dois problemas principais com a coleta de dados. Primeiro, a coleta de dados não é possível no projeto e desenvolvimento do sistema; o sistema não é monitorável se ele não é operacional. Segundo, a monitoração de muitos sistemas é uma atividade complexa, a qual envolve consideráveis custos humanos e computacionais (Sauer & Chandy, 1981).

A coleta de dados pode ser utilizada para obter dados de um sistema existente, com o objetivo de comparar valores obtidos, com os valores fornecidos por um modelo do mesmo sistema. Assim, o sistema pode ser modelado e resolvido por simulação ou métodos analíticos e as entradas para esse modelo podem ser abstraídas dos dados coletados no sistema em operação.

A coleta de dados pode ser realizada por meio de dois recursos:

- **Monitores de Hardware:** é empregado um hardware específico para coletar e analisar dados pertinentes ao objeto em estudo, isto é, coletar informações sobre o uso de um dado recurso e também informações sobre várias atividades, sem perturbar o sis-

tema. Os monitores de hardware devem limitar-se a coletar os dados e não alterá-los, mantendo fidelidade aos dados originais;

- Monitores de software: são usados para monitorar características específicas de software, como exemplo, verificar a existência ou não de uma fila de espera associada a um recurso do sistema.

Benchmarks

Um *benchmark* é um programa o qual mede o desempenho de um computador, de uma parte do mesmo ou de um outro programa os quais executam sempre a mesma tarefa de tempos em tempos (Intel, 2004).

Os *benchmarks* são a abordagem mais utilizada para a avaliação de desempenho. Eles podem ser utilizados para: comparar o desempenho de máquinas diferentes, reprojeter hardware e software, ajudar na otimização de programas e previsão de desempenho de aplicações em computadores específicos. A avaliação de um único componente do sistema, não permite a avaliação global do sistema. Os *benchmarks* permitem avaliar o sistema como um todo, visto que, a execução de um programa real avalia todos os aspectos necessários, incluindo aqueles relacionados com arquitetura do sistema, eficiência do compilador, sobrecarga do sistema operacional, sobrecarga de comunicação, identificam os pontos fortes e fracos de um sistema em particular ou de aplicações, etc. (Couto, 1999). Os *benchmarks* são ferramentas úteis, principalmente para a comparação de sistemas.

Construção de Protótipos

A prototipação é uma versão simplificada de um sistema computacional, que contém, as características consideradas relevantes para a análise do sistema. Os protótipos apresentam um menor custo e uma maior facilidade de alteração em relação à construção e alteração do sistema real (Orlandi, 1995), mas, um custo maior em relação as outras técnicas de aferição. Porém a dificuldade em se construir protótipos está na determinação das características essenciais do sistema.

O desenvolvimento de um protótipo pode ser dividido em seis passos (Santana *et al.*, 1994):

1. Avaliação da adequabilidade da utilização do protótipo;
2. Avaliação dos domínios funcionais e comportamentais do sistema e das características principais que devem ser consideradas;

3. Desenvolvimento do protótipo;
4. Teste do protótipo;
5. Após o passo 4, o protótipo já provê uma idéia do sistema real. Nesse instante o desenvolvedor pode inferir sobre as características fundamentais do sistema e, se for o caso, fazer modificações no protótipo;
6. Os passos 4 e 5 são repetidos iterativamente até que se tenha toda a funcionalidade do sistema representado no protótipo.

2.2.2 Técnicas de Modelagem

As técnicas de modelagem consistem na representação de um determinado sistema em forma de um modelo, visando a obtenção de informações que auxiliem na resolução de questões sobre o sistema real. Um modelo nada mais é do que uma descrição de um determinado sistema, tendo-se a preocupação de abstrair suas características relevantes (Soares, 1990).

A modelagem, no contexto de avaliação de desempenho, deve contemplar as características essenciais de um sistema real, sendo que, com a solução desse modelo, pode-se ter uma aproximação de como o sistema se comportaria se fosse efetivado (Francès, 2001).

Processo de Modelagem

Modelagem é um conjunto de etapas independentes, entretanto, intimamente inter-relacionadas. Na Figura 2.1 são apresentadas as etapas seguidas em um processo de modelagem (Francès, 2001):

Na fase inicial da modelagem, deve-se criar uma especificação condizente com o sistema real, na qual devem estar contidos os componentes do sistema relevantes à avaliação e a relação entre eles. Uma questão importante, para o desenvolvimento de um modelo, é definir qual especificação será utilizada para a representação do modelo. Três técnicas têm sido amplamente usadas, possuindo suas vantagens e desvantagens conforme o domínio de aplicação considerado: Redes de Filas, Redes de Petri e *Statecharts* (Santana *et al.*, 1994).

Após a confecção do modelo, com uma das técnicas citadas, deve-se parametrizá-lo com elementos que serão dados de entrada para a próxima fase (solução). Uma vez obtido o modelo, pode-se solucioná-lo de duas maneiras distintas: para simulação ou para solução analítica. Por fim, a modelagem deve apresentar seus resultados, utilizando-se de uma maneira conveniente: gráficos ou arquivos-texto são, geralmente, utilizados nessa fase (Francès, 2001).

Figura 2.1: Etapas do Processo de Modelagem

Na seção 2.2.3 serão discutidas as formas tradicionais para a especificação de modelos e na seção 2.2.4 as formas para solucionar os modelos especificados.

2.2.3 Técnicas para Especificação de Modelos

Redes de Filas

As redes de filas (RF) possuem uma base matemática bastante sólida, contudo, sua representação gráfica oferece apenas os elementos fila e servidor, o que em muitos casos pode não ser suficiente. Em alguns sistemas, desejam-se representar situações que não constituem necessariamente uma fila ou um servidor. Por exemplo, um processo em um computador pode se encontrar em três possíveis situações: processando, bloqueado ou pronto. Essas situações são estados abstratos que, por não consistir em filas e servidores, não são tratadas apropriadamente em redes de filas. Além disso, filas e servidores necessitam de uma representação mais minuciosa, para que se possa especificar as várias situações, que, tanto a fila (uma fila pode estar vazia ou não), quanto o servidor (um servidor pode estar livre ou ocupado), possam se encontrar em um determinado instante.

Redes de Petri

As redes de Petri (RP) possuem características bastante interessantes em sua representação. Uma das mais interessantes é a possibilidade da individualização de clientes, recurso que não é usual na maioria das técnicas. Há situações em que essa característica

é primordial, por exemplo, na migração de processos para balanceamento de carga entre máquinas, pois se pode desejar migrar determinado processo, e por isso deve-se saber onde ele se encontra. Entretanto, há uma certa dificuldade na representação de processos paralelos, mesmo utilizando-se da rede de distribuição (rede elementar para representar criação de processos paralelos). Esse aspecto, piora à medida que o modelo cresce, o que aliás é outro aspecto a ser ponderado. As redes originais, por não possuírem nenhum mecanismo de hierarquia, tendem a fazer com que os modelos cresçam substancialmente, quando a complexidade dos sistemas a serem representados aumenta.

Statecharts

Statecharts é uma técnica de representação de sistemas por intermédio da visão de seus estados e a modificação deles em consequência da ocorrência de uma determinada interferência. Essa técnica é uma extensão das máquinas de estado finito, que possibilita a representação de hierarquia, concorrência e comunicação entre os diversos estados de um determinado sistema (Harel, 1987).

Na representação *Statecharts*, a forma explícita de algumas características (como o paralelismo e a hierarquia entre estados) é um atrativo considerável. Essa peculiaridade permite que relacionamentos complexos entre componentes de um determinado sistema sejam mostrados de maneira mais efetiva, o que não é contemplado na maioria das técnicas (Harel, 1987). Entretanto, os *Statecharts* não contemplam a descrição explícita do caminho linear seguido por um determinado cliente (que é claro em RF e RP), quando há a presença de estados ortogonais, pois podem haver vários subestados ativos em um determinado momento, o que não significa que todos os clientes (ou um cliente em particular) tenham seguido por aquela trajetória. Além disso, a individualização de clientes (como feito com a utilização dos tokens em RP), fundamental em certas situações (tal como em escalonamento de processos, em que se pode decidir por migrar um processo em particular), não é implementada em *statecharts*.

2.2.4 Soluções para o Modelo

Após proceder-se à escolha da técnica utilizada para a representação do modelo, deve-se decidir qual a solução a ser dada ao mesmo. Existem, basicamente, duas técnicas de solução disponíveis: a analítica e a simulação (Aries *et al.*, 2002). As duas técnicas têm suas vantagens e desvantagens, que normalmente são os fatores determinantes na escolha de uma em detrimento da outra (Francès *et al.*, 2001).

Solução Analítica

Os modelos analíticos descrevem a operação do sistema e sua carga em termos completamente abstratos. As estimativas de desempenho são obtidas por meio da resolução analítica ou numérica do modelo matemático resultante.

O método analítico é geralmente mais rápido e, por isso, o preferido (Macfadyen, 2003); porém, nem sempre ele é aplicável. Muitas vezes, é necessário fazer um certo número de simplificações para tornar possível a solução do modelo (Aries *et al.*, 2002), (Santana *et al.*, 1994). As simplificações introduzidas podem fazer com que o modelo não seja uma representação fiel do sistema real, fato que pode simplesmente tornar sem sentido a utilização do modelo (Santana *et al.*, 1994).

Para exemplificar a dificuldade imposta por certas restrições, na solução analítica de modelos baseados em redes de filas, não se pode estabelecer prioridades para as disciplinas de filas (Soares, 1990). Isto é, todos os usuários de um determinado recurso possuem a mesma prioridade, o que inviabilizaria, por exemplo, a representação do esquema de prioridades atribuídas aos processos no sistema UNIX. Ainda há algumas outras restrições impostas à solução analítica, como no exemplo, a utilização de certas distribuições de probabilidade que possuem um tratamento matemático mais factível, em certas situações, pode ser um fator limitante.

Nas situações em que várias simplificações (restrições) comprometem a exatidão dos resultados do modelo, pode ser mais conveniente adotar a solução por simulação.

Solução por Simulação

Se o modelo proposto para o sistema envolve grande gama de informações e/ou exige que não se façam algumas das simplificações vistas na seção anterior, o modelador pode optar por resolver o modelo utilizando a simulação (Singleton, 2002). A simulação é utilizada para solucionar modelos que tentam imitar o comportamento do sistema, constituindo-se em um experimento estatístico que observa o comportamento do modelo no decorrer do tempo. A partir de um programa computacional, pode-se exercitar o modelo de alguma maneira. Isso pode ser realizado por meio de dados obtidos de medidas do sistema ou por valores aleatórios gerados a partir de distribuição, com a finalidade de representar as variáveis do sistema (Soares, 1990).

Na análise de desempenho de sistemas computacionais, a simulação é uma técnica bastante útil, especialmente, se o sistema a ser caracterizado não estiver disponível. Modelos de simulação são programas de computador, nos quais a operação de um sistema e sua carga são descritas utilizando-se algoritmos apropriados.

2.3 Considerações Finais

A avaliação de desempenho dos sistemas computacionais distribuídos dá uma idéia de como estes sistemas se apresentam. A escolha da técnica a ser utilizada na avaliação de desempenho, seja ela técnica de aferição ou de modelagem, corresponde a uma etapa de fundamental importância dentro do processo de avaliação de desempenho. Esta escolha está diretamente relacionada ao estado de desenvolvimento do sistema que se deseja avaliar, quais questões serão respondidas, a complexidade do sistema, o quão rápido os resultados são requeridos (Singleton, 2002).

As técnicas de aferição são indicadas quando o sistema a ser avaliado já existe ou está em fase final de desenvolvimento. A construção de protótipos é mais indicada para a avaliação de desempenho de sistemas que se encontram na fase final de desenvolvimento. Os *benchmarks* têm o seu uso preferencial para decidir sobre a seleção de novos sistemas. A coleta de dados é a técnica de aferição que oferece os resultados mais precisos. Esta técnica é recomendada para os casos em que o sistema já exista. Porém, quando o sistema não existir, ou quando a aferição irá perturbar o funcionamento do sistema ou quando é difícil incluir um monitor no sistema sem que haja a influência da própria aferição no resultado da avaliação, tem-se a opção de se tomar uma abstração do sistema, levando-se em consideração as características essenciais. Essas características farão parte do modelo, a partir do qual são realizadas certas inferências, sem a necessidade de realizar-se a efetiva experimentação do sistema.

Dentre as técnicas para representação de modelos existentes pode-se destacar as redes de filas, redes de Petri e Statecharts. A técnica de redes de filas é bastante utilizada para avaliação de sistemas computacionais, pois estuda o fenômeno da formação de filas.

Após a escolha da técnica para representação de modelos, deve-se analisar qual a solução do modelo a ser empregada: analítica ou simulação. Para modelos menos complexos é recomendável a utilização de soluções analíticas, pois podem admitir certas simplificações no modelo. Porém, em sistemas mais complexos, é recomendável o uso de simulação. Pois, a simulação é uma técnica que considera um modelo mais apurado do sistema, sendo possível especificar mais detalhes dentro do modelo. A Figura 2.2 apresenta o inter-relacionamento entre as duas soluções de um modelo, assim como as variantes que determinam quando, preferencialmente, utilizar uma solução em detrimento da outra (Francès, 2001).

A modelagem analítica trata da representação matemática de um sistema (Macedo & Saad, 1979), geralmente os modelos são representados utilizando-se redes de filas. Essa técnica é geralmente mais rápida, mas, nem sempre aplicável (Kleinrock, 1976). Ela fornece resultados precisos, porém há casos em que o modelo pode-se tornar muito complexo. Nes-

Figura 2.2: Uso Preferencial das Técnicas de Modelagem

ses casos, as simplificações colocadas no modelo para que seja possível sua solução analítica tendem a reduzir sua validade (Macedo & Saad, 1979). Nas situações em que várias simplificações comprometem a exatidão dos resultados do modelo, pode ser mais conveniente adotar a solução por simulação (Frances, 2001).

A avaliação de desempenho de sistemas computacionais distribuídos é uma área em evolução, uma vez que os usuários estão cada vez mais exigentes em relação aos serviços que lhes são prestados pelos sistemas. O que torna o processo de avaliação de desempenho uma prática recomendável em todos os sistemas distribuídos, contribuindo para a evolução dos mesmos.

Representação dos Modelos

3.1 Considerações Iniciais

A representação de modelos visando a avaliação de desempenho é realizada por meio de técnicas de modelagem, que criam uma abstração do sistema a ser modelado, conservando as características essenciais do sistema. Para representar o modelo de um sistema, pode-se fazer uso de redes de filas, redes de Petri e Statecharts, apresentadas no Capítulo 2.

Este capítulo aborda de maneira mais abrangente as técnicas mencionadas e apresenta duas novas abordagens formais para a especificação de modelos de desempenho: *Statecharts* Estocásticos (especificação para sistemas de filas baseados em Statecharts) e *Queuing Statecharts* (uma combinação entre as representações de redes de filas e *Statecharts*).

3.2 Redes de Filas

Os sistemas de filas têm sido estudados desde o começo dos sistemas de telefonia por A. K. Erlang. A partir da década de 60, observa-se a aplicação, na indústria da computação, de modelos de redes de filas, utilizados para representar e analisar o desempenho de sistemas computacionais, de recursos compartilhados, de comunicação e de produção (Balsamo *et al.*, 1998). A utilização de redes de filas é fundamental para o desenvolvimento de pacotes de softwares visando o projeto, avaliação e melhoria dos sistemas computacionais.

Qualquer sistema que consiste na chegada de requisições de serviço para um ou mais recursos de capacidade finita pode ser denominado sistema de filas. Em sistemas computacionais é comum a formação de filas. Deste modo, há filas de requisições esperando para serem processadas, filas de requisições de base de dados, filas de requisições de I/O (*input/output*),

etc. (Allen, 1990).

Uma rede de filas é formada por várias filas que se interconectam entre si de modo que o usuário ao sair de uma fila e utilizar um determinado recurso pode (com certa probabilidade) dirigir-se a outra (Magalhães, 1996). Aos usuários da fila costuma-se chamar de clientes e ao recurso de servidor. O conjunto de filas de clientes e servidor é denominado centro de serviço. A Figura 3.1 apresenta a notação gráfica para centro de serviço (Ajmone Marsan *et al.*, 1986).

Figura 3.1: Centro de Serviço

3.2.1 Notação para Sistemas de Filas

Para especificar um sistema de filas é necessário descrever seis parâmetros (Jain, 1991). Uma notação bastante aceita para redes de filas é a notação de Kendall (Jain, 1991). A notação de Kendall obedece ao seguinte formato:

$$\mathbf{A}/\mathbf{B}/\mathbf{c}/\mathbf{K}/\mathbf{m}/\mathbf{Z}$$

Onde:

- **A**: tipo de distribuição da variável aleatória que define a taxa de chegada de clientes (padrão de chegada);
- **B**: tipo de distribuição da variável aleatória que define a taxa de prestação de serviço (padrão de serviço);
- **c**: número de servidores da estação servidora ($c \in \{1, 2, \dots, n\}$);
- **K**: capacidade da fila - número de clientes que podem estar simultaneamente aguardando na estação servidora ($K \in \{1, 2, \dots, n\}$) - quando omitida, $K = +\infty$;
- **m**: população máxima de usuários da rede de filas da qual a estação em questão faz parte, quando omitida, $m = +\infty$;

- **Z**: disciplina de serviço da fila, definido por um dos algoritmos de escalonamento apresentados a seguir:
 - FCFS (*First Come - First Server*): o primeiro cliente que chega é o primeiro cliente a ser atendido;
 - LCFS (*Last Come - First Served*): o último cliente que chega é o primeiro cliente a ser atendido;
 - RR (*Round Robin*): o cliente escalonado é atendido por um pequeno intervalo de tempo - *quantum*. Se o *quantum* não for suficiente o cliente é colocado no final da fila e, oportunamente, retorna ao servidor até que o serviço requisitado seja completado.
 - Prioridades e Preempção: as filas podem requerer a atribuição de prioridades para os clientes. As prioridades podem ser preemptivas, isto é, o atendimento de um cliente é interrompido caso chegue um cliente com maior prioridade, ou não preemptivas, ou seja, o cliente em atendimento não é afetado, porém o próximo a ser atendido será sempre o de maior prioridade (Santana *et al.*, 1994).

A omissão de K e Z na representação da fila indica que a capacidade da fila é infinita e a disciplina da fila é FCFS.

A próxima seção apresenta uma breve descrição de algumas distribuições utilizadas na modelagem de sistemas.

Distribuição de Probabilidades

Uma distribuição de probabilidade descreve o comportamento de uma variável aleatória no decorrer do tempo. Uma variável aleatória é uma regra de associação de um valor numérico a cada ponto de um espaço amostral. Em um sistema de filas, a distribuição de probabilidades representa o padrão de prestação de serviços a cada cliente e o padrão de chegada de clientes em um centro de serviço.

A seguir é feita uma breve descrição de algumas distribuições importantes na modelagem de sistemas (Soares, 1990).

- **Distribuição Uniforme** - especifica que os valores compreendidos entre um máximo e um mínimo são equiprováveis. A distribuição uniforme geralmente é usada quando não se tem maiores informações sobre a variável aleatória, somente os seus limites são conhecidos.

- **Distribuição Exponencial** - é a distribuição cuja probabilidade de acontecer um evento em um pequeno intervalo de tempo é proporcional ao tamanho desse intervalo. Se a propriedade de ocorrência de mais de um evento nesse intervalo é nula, e se a ocorrência de um evento é independente da ocorrência de outros, então o intervalo entre ocorrências tem uma distribuição exponencial. Uma propriedade interessante dessa distribuição é que ela não tem memória (*memoryless*), isto é, o tempo de serviço destinado a um cliente independe do tempo de serviço gasto com clientes anteriores (Jain, 1991).
- **Distribuição de Erlang** - é a distribuição derivada da soma de um número inteiro de variáveis aleatórias independentes e exponencialmente distribuídas. Essa distribuição é utilizada em teoria de filas, quando uma atividade ou tempo de serviço ocorre em fases, com cada fase sendo exponencialmente distribuída.
- **Distribuição Hiperexponencial** - é um caso particular de distribuição exponencial que admite uma variância muito grande em relação à média. Essa distribuição pode ser utilizada para descrever o tempo de serviço que um servidor dispensa a um cliente, pois os tempos de atendimento podem ser muito discrepantes em relação às suas média.
- **Distribuição de Poisson** - é a distribuição discreta que estabelece que se a duração do intervalo de tempo entre chegadas é exponencialmente distribuído e se as chegadas ocorrem uma de cada vez, então o número de chegadas em um intervalo fixo qualquer possui uma distribuição de Poisson.

3.2.2 Classificação das Redes de Filas

As redes de filas podem ser classificadas de acordo com o fluxo de clientes na rede, os tipos de clientes e a capacidade de absorção dos centros de serviço. Observando o fluxo de clientes na rede pode-se dizer se ela é uma rede aberta ou fechada. Uma rede aberta é caracterizada pela variação do número total de clientes na rede, isto é, pode ocorrer tanto a entrada de novos clientes quanto a saída de clientes da rede. Já em uma rede fechada há um número fixo de clientes circulando pelo sistema. A Figura 3.2 apresenta o esquema de redes abertas e fechadas.

Figura 3.2: Redes Abertas e Fechadas

Os tipos de clientes de uma rede de filas determinam se a rede é monoclasse ou multiclasse. Em uma rede de filas monoclasse todos os clientes são iguais e recebem o mesmo tratamento. Nas redes de filas multiclasse os clientes têm tratamento diferenciado, de acordo com a classe a que pertençam.

Os centros de serviços podem ser classificados de acordo com a capacidade de cada centro em absorver os seus clientes. Os centros podem ser de capacidade fixa ou centros *delay*. Os centros de serviço de capacidade fixa são aqueles em que os clientes competem pelos recursos. Os centros *delay* possuem um número muito grande de servidores disponíveis. A Figura 3.3 apresenta os tipos de absorção de centros de serviço.

Figura 3.3: Tipos de Absorção de Centros de Serviço

A Figura 3.4 apresenta a especificação de um servidor de arquivos em redes de filas (Francês *et al.*, 2000a). O sistema de arquivos consiste de um processador. Toda vez que o processador está ocupado, a requisição para usar o processador é colocada na fila do processador. Depois que uma requisição é processada, ela pode deixar o sistema ou usar o disco. No centro de serviço do disco, a requisição pode ser colocada na fila do disco se o disco estiver em uso. Depois que a requisição terminou de usar o serviço do disco, ela retorna para o processador.

Figura 3.4: Servidor de Arquivos Representado em Rede de Filas

3.2.3 Parâmetros de Desempenho e Restrições em Redes de Filas

Dois fatores formam a essência da teoria das filas: o padrão de chegada dos clientes (taxa em que os clientes chegam a uma fila) e o padrão de serviço nos centros (taxa de serviço

de um servidor) (Kleinrock, 1976). Através desses fatores é possível avaliar a utilização de um servidor (percentual que um servidor está ocupado), o tempo de residência (medida do tempo que os clientes levam para completar o trabalho), o tamanho médio da fila (número médio de clientes na fila de um centro de serviço), etc. (Santana *et al.*, 1994).

Algumas restrições são impostas à solução analítica, quando se utilizam redes de filas. As redes de filas não permitem a posse simultânea de recursos, porém pode ocorrer situações em que se deseja que um determinado cliente usufrua de mais de um recurso ao mesmo tempo. Este tipo de restrição inviabiliza, por exemplo, a transmissão de uma mesma mensagem, simultaneamente, através de duas linhas de comunicação.

A disciplina das filas com prioridades não são permitidas em redes de filas. Isto impede, por exemplo, a resolução de um modelo de um sistema operacional como o UNIX, que possui um esquema de prioridades. As situações que não constituem filas ou servidores (estados abstratos) não são tratadas adequadamente em redes de filas.

Essas são algumas das restrições impostas à solução analítica de um modelo baseado em redes de filas. Entretanto, muitas vezes, tais restrições são violadas em sistemas reais e a solução analítica tem o seu campo de validade restringido.

3.3 Redes de Petri

O conceito de redes de Petri teve origem na dissertação de Carl Adam Petri, em 1962 na faculdade de Matemática e Física da Universidade de Darmstadt na Alemanha Ocidental (Petri, 1966). O trabalho de Petri chamou a atenção de A. W. Holt, que junto com outros pesquisadores desenvolveu muito da teoria, notação e representação das redes de Petri (Murata, 1989).

Redes de Petri é uma ferramenta de modelagem efetiva para a descrição e a análise de concorrência e sincronização em sistemas paralelos, demonstrando as ações cooperativas de diferentes entidades. Ela permite a verificação de propriedades e a correção do sistema especificado (Maciel *et al.*, 1996). A utilização das redes de Petri permite modelar sistemas concorrentes, paralelos, distribuídos, assíncronos, não-determinísticos e estocásticos (Francès *et al.*, 2000b).

A estrutura padrão de uma rede de Petri é um grafo bipartido que compreende um conjunto de lugares, transições e arcos (Ajmone Marsan *et al.*, 1986) (Figura 3.5) e regras que governam a operação da rede. Os lugares representam a localização onde objetos esperam por processamento. A localização pode ser física (por exemplo, uma fila onde uma mensagem espera para ser processada) ou um estado (por exemplo, um recurso ocioso). As transições

representam processos ou eventos. Os arcos conectam lugares a transições e transições a lugares; a direção do caminho é indicada por uma seta no final do arco. (Moore & Breiman, 1995). Os arcos podem ser únicos ou múltiplos.

Figura 3.5: Descrição dos Elementos Básicos de Redes de Petri

A definição formal de Redes de Petri é dada por:

- $PN = (P, T, A)$,
- $P = \{p_1, p_2, \dots, p_n\}$,
- $T = \{t_1, t_2, \dots, t_n\}$,
- $A_i \subset (P \times T)$,
- $A_0 \subset (T \times P)$,
- $A = (A_i \cup A_0)$.

Um exemplo de uma representação gráfica de uma rede de Petri é mostrado na Figura 3.6 (Ajmone Marsan *et al.*, 1986). A rede de Petri descrita na Figura 3.6 contém cinco lugares (p_1, p_2, p_3, p_4, p_5) e cinco transições (t_1, t_2, t_3, t_4, t_5).

Figura 3.6: Exemplo de Representação Gráfica de uma Rede de Petri

3.3.1 Redes de Petri Marcadas

Os lugares podem conter *tokens*. *Tokens* são usados para simular o dinamismo e a concorrência das atividades do sistema (Murata, 1989). Uma rede de Petri com *tokens* é chamada rede de Petri marcada. O estado de uma rede de Petri marcada é definido pelo número de *tokens* (m_i) que cada lugar (p_i) contém. O estado da rede de Petri é denotado por $M = m_1, m_2, \dots, m_n$.

A definição formal de uma rede de Petri marcada é dada por:

- $PN = (P, T, A, M_0)$,
- $P = \{p_1, p_2, \dots, p_n\}$,
- $T = \{t_1, t_2, \dots, t_m\}$,
- $A_i \subset (P \times T)$,
- $A_0 \subset (T \times P)$,
- $A = (A_i \cup A_0)$,
- $M_0 = m_{01}, m_{02}, \dots, m_{0n}$, onde m_{0i} denota o número de *tokens* no lugar p_i na marcação inicial da rede (Ajmone Marsan *et al.*, 1986).

O comportamento dos sistemas pode ser descrito em função dos seus estados e suas alterações. Para simular o comportamento dinâmico dos sistemas a marcação da rede de Petri é modificada a cada ação realizada (transição disparada), segundo algumas regras de execução (Maciel *et al.*, 1996).

As regras de execução das redes de Petri especificam as condições sob as quais os processos ou eventos podem ocorrer. Três regras governam o disparo das transições (Moore & Breman, 1995), (Ajmone Marsan *et al.*, 1986).

1. Quando todos os lugares anteriores às transições são ocupados por um *token*, a transição é habilitada.
2. Uma vez habilitada, a transição dispara.
3. Quando a transição dispara, exatamente um *token* é removido de cada lugar anterior e exatamente um *token* é colocado em cada lugar posterior às transições.

Figura 3.7: Disparo de Transição

A Figura 3.7 mostra estas regras para uma transição com dois lugares anteriores (A, B) e dois lugares posteriores (C, D) (Moore & Breman, 1995).

Existem algumas redes básicas, através das quais podem ser implementadas todas as demais. Essas redes são chamadas de elementares. Na próxima seção são apresentadas as principais.

3.3.2 Redes de Petri Elementares

Nesta seção são apresentadas algumas redes básicas e essenciais para a modelagem de sistemas em geral, a partir das quais, sistemas mais complexos podem ser modelados (Maciel *et al.*, 1996).

Seqüenciamento

O seqüenciamento é uma rede que representa a execução de uma ação, desde que uma condição seja verdadeira. Após a execução desta ação tem-se uma nova condição que poderá possibilitar a execução de uma nova ação (Figura 3.8).

Figura 3.8: Seqüenciamento

Distribuição

A distribuição possibilita a criação de processos paralelos a partir de um processo pai. As novas condições permitem a execução de outras tarefas paralelamente, ou seja, p_0 é pré-condição para a execução de uma tarefa e p_2 é pré-condição para a execução da tarefa p_1 . Na Figura 3.9 é apresentada a distribuição.

Figura 3.9: Distribuição

Junção

A junção é a rede que possibilita a sincronização entre processos. Ela é de fundamental importância para a modelagem de atividades concorrentes. A junção é apresentada na Figura 3.10.

Figura 3.10: Junção

Escolha Não-Determinística

Escolha não-determinística é quando em uma rede de Petri têm-se duas transições com um mesmo lugar de entrada. Neste modelo, o disparo de uma transição inabilita o disparo de outra transição. A Figura 3.11 apresenta a escolha não-determinística do disparo de transições (Nonato, 1999).

Figura 3.11: Escolha Não-Determinística

3.3.3 Redes de Petri Coloridas

As redes de Petri coloridas têm como objetivo principal a redução do tamanho do modelo, permitindo que tokens individualizados (cores) representem diferentes processos ou recursos em uma mesma rede (Maciel *et al.*, 1996). As cores não significam apenas cores ou padrões. Elas podem representar tipos de dados complexos, usando a nomenclatura de colorida apenas para referenciar a possibilidade de distinção entre os *tokens* (Jensen *et al.*, 1990). A Figura 3.12 apresenta uma rede colorida em que os arcos são rotulados com cores (DNS, Telnet, FTP e HTTP)¹. Há um cliente TCP/IP que pode solicitar quatro tipos de serviços a quatro servidores diferentes (o servidor DNS, o servidor Telnet, o servidor FTP e o servidor HTTP). As solicitações do cliente devem ser diferenciadas, de acordo com o servidor que vai atendê-las. A associação de cores e marcas individualizadas para cada requisição e as restrições colocadas nos arcos determinam quais marcas podem atravessar esses arcos. Assim, o *token* se destinará ao arco cuja cor for idêntica à da marca. Pode-se perceber que os *tokens* de “Cliente” só habilitarão as transições t_0 e t_3 (Francés, 2001), pois os *tokens* que ele possui habilitam somente essas transições. O cliente só pode solicitar serviços de DNS e HTTP.

3.3.4 Redes de Petri Estocásticas

A união das características de redes de Petri (que são voltadas para descrever situações de concorrência e sincronismo) com um modelo estocástico permite realizar o estudo de desempenho de sistemas computacionais. Assim, as redes de Petri estocásticas associam a cada transição uma variável aleatória exponencialmente distribuída (Ajmone Marsan *et al.*, 1998), que expressa a quantidade de tempo decorrido até que a transição seja habilitada (Francés, 2001).

As redes de Petri estocásticas podem ser formalmente definidas como sendo

¹DNS (Domain Name System); FTP (File Transfer Protocol); HTTP (Hyper Text Transfer Protocol)

Figura 3.12: Rede de Petri Colorida de um Cliente TCP/IP

(Ajmone Marsan *et al.*, 1986):

- $SPN = (P, T, A, M_0, L)$,
- $P = \{p_1, p_2, \dots, p_n\}$,
- $T = \{t_1, t_2, \dots, t_n\}$,
- $A_i \subset (P \times T)$,
- $A_0 \subset (T \times P)$,
- $A = (A_i \cup A_0)$,
- $M_0 = m_{01}, m_{02}, \dots, m_{0n}$,
- $L = l_1, l_2, \dots, l_m$, onde L é o conjunto de taxas de disparo associadas às transições.

As redes de Petri estocásticas permitem a obtenção de estatísticas como, por exemplo, o tempo médio que um *token* gasta para percorrer a rede, o número médio de vezes que uma determinada transição dispara e o número médio de *tokens* em um lugar.

3.3.5 Redes de Petri Estocásticas Generalizadas

Em determinadas situações não é desejável associar um tempo aleatório a cada transição, mas sim apenas àquelas que estão relacionadas com eventos que exerçam maior impacto no desempenho do sistema em estudo. As redes de Petri estocásticas generalizadas (GSPN -

Generalized Stochastic Petri Nets) permitem transições pertencentes a duas diferentes classes: transições imediatas e transições temporizadas (Ajmone Marsan *et al.*, 1986).

As transições imediatas são disparadas em tempo zero uma vez que elas são habilitadas. As transições temporizadas disparam após um tempo aleatório exponencialmente distribuído. A Figura 3.14 mostra a representação gráfica para as transições imediatas e temporizadas.

Figura 3.13: Representação Gráfica de Transições

A definição formal de uma rede de Petri estocástica generalizada é representada por:

- $GSPN = (P, T, A, M_0, L)$,
- $P = \{p_1, p_2, \dots, p_n\}$,
- $T = \{t_1, t_2, \dots, t_n\}$,
- $A_i \subset (P \times T)$,
- $A_0 \subset (T \times P)$,
- $A = (A_i \cup A_0)$,
- $M_0 = m_{01}, m_{02}, \dots, m_{0n}$.
- $L = l_1, l_2, \dots, l'_m$ onde m' é o número de transições temporizadas.

A Figura 3.14 mostra a representação gráfica de um servidor de arquivos descrito na seção 3.2.2 , agora utilizando redes de Petri.

3.3.6 Vantagens e Desvantagens de Redes de Petri

As redes de Petri constituem uma técnica poderosa devido ao fato de serem voltadas para a especificação de sistemas concorrentes. Elas permitem que os lugares sejam genéricos, isto é, eles podem ser tanto recursos de um sistema (por exemplo, um processador), quanto situações abstratas (como recurso ocupado). Elas possibilitam a representação inicial do sistema através da marcação inicial. E por meio de *tokens* têm-se a situação corrente do sistema. Usando redes de Petri coloridas os *tokens* podem ser individualizados.

Figura 3.14: Servidor de Arquivos em GSPN

As redes de Petri não possuem uma representação para tratar filas, pois um lugar com vários *tokens* não pode ser encarado como um fila (uma fila deve ter um algoritmo de escalonamento, uma taxa de chegada e, possivelmente, prioridades para clientes). Redes de Petri não possuem uma representação explícita de paralelismo. Pode ocorrer uma explosão no número de lugares e transições do modelo, uma vez que um lugar não pode ser subdividido em sublugares (Francès *et al.*, 2000b).

3.4 Statecharts

Os *Statecharts* foram propostos por Harel (Harel, 1987) e constituem uma técnica de representação de sistemas através da visão de seus estados e a modificação deles em consequência da ocorrência de uma determinada interferência. *Statecharts* é uma extensão do diagrama tradicional de estados-transições aos quais foram adicionados conceitos como hierarquia entre estados (*depth*), ortogonalidade (representação de atividades paralelas) e interdependência entre estados (mecanismos de comunicação) (Harel, 1987).

Em (Harel, 1987), *Statecharts* são definidos como:

Statecharts = diagrama de estados + hierarquia + ortogonalidade + mecanismos de comunicação.

3.4.1 Definição e Notação Básica

A fundamentação de *Statecharts* é baseada nos seguintes elementos básicos: estados, transições, eventos, condições, ações, expressões, rótulos e variáveis.

Os estados representam valores das variáveis do sistema em um determinado instante. Eles podem ser básicos ou não-básicos. Os estados básicos são aqueles que não possuem subestados e os estados não-básicos são decompostos em subestados. Essa decomposição pode ser de dois tipos: OR ou AND. Se a decomposição é do tipo OR, então o sistema sempre estará em um único subestado em um certo instante. Porém, se a decomposição é do tipo AND, o estado estará em mais de um subestado, simultaneamente (Francès, 2001).

Evento é a entidade que causa uma interferência no comportamento atual do sistema, levando esse sistema a outro comportamento. Opcionalmente, a um evento pode ser adicionado uma condição que garante que o evento só ocorrerá se satisfeita aquela determinada condição.

As ações são consideradas para representar os efeitos do paralelismo em *Statecharts*. Ações podem ser uma mudança de expressão, uma mudança de uma variável ou eventos que são disparados em outros componentes paralelos.

As transições são representações gráficas usadas para denotar uma mudança de estado dentro do sistema. A transição pode ocorrer em função de um estímulo externo, mas também da ocorrência de uma condição verdadeira. Rótulos podem ser acrescentados às setas para prover algum significado adicional (Francès, 2001).

Os *Statecharts* possuem outros recursos que ampliam o seu poder de especificação. Alguns desses recursos são mostrados a seguir.

3.4.2 Estado Default

Para representar um estado inicial, *Statecharts* provê o conceito de estado *default*. A Figura 3.15 apresenta a representação gráfica do estado *default*. O estado *default* de D é o estado A. Isso significa que o estado D sempre começa pelo seu subestado A.

Figura 3.15: Estado *Default*

3.4.3 Entrada por Condição

Os *Statecharts* permitem a especificação da escolha de um estado dentre vários, através de uma condição preestabelecida. A Figura 3.16 apresenta uma entrada por condição (Harel, 1987).

Figura 3.16: Entrada por Condição

3.4.4 Delays e Timeouts

A expressão *timeout (event, number)* indica que o sistema só irá transicionar para outro estado após decorrido o número de unidades de tempo (*number*) da ocorrência do evento especificado na expressão. A Figura 3.17 (Harel, 1987) mostra a notação para o *delay*, que indica que se deve esperar dois segundos para realizar um determinado evento.

Figura 3.17: Estado com *Delay* de 2 Segundos

3.4.5 Estados Parametrizados

Em muitos casos, diferentes estados possuem uma estrutura interna idêntica. Nessas situações é melhor visualizá-los como um estado simples com parâmetros (Harel, 1987). A Figura 3.18 (Francès, 2001) apresenta um sistema com dezesseis conjuntos processador/disco (*processor/disc*), utilizando a especificação de estados AND parametrizados.

Figura 3.18: Estados Parametrizados Processor/Disc

3.4.6 Semântica dos Statecharts

A semântica Statecharts se baseia em uma sequência de instantes de tempo σ_i $i \geq 0$, correspondente à taxa de execução do sistema. Por definição (Harel, 1987), existe um conjunto de intervalos de tempos definidos por $I = (\sigma_i, \sigma_{i+1})$, onde cada $\Delta\sigma(\sigma_{i+1} - \sigma_i)$ representa o tempo dispensado a um determinado passo i . O sistema irá reagir a cada final de intervalo $\sigma \rightarrow \sigma_{i+1}$, ou seja, em σ_{i+1} o sistema reagirá, apresentando uma nova configuração de estados (Francès, 2001).

Um estímulo externo, em σ_{i+1} , é uma tripla (Π, θ, ξ) , onde Π é um conjunto de eventos primitivos externos que ocorrem em I_i , θ é o conjunto de condições primitivas externas, cujo valor é verdadeiro (true) em (σ_i, σ_{i+1}) , e ξ é uma função determinada pelo ambiente externo, de maneira que, para uma variável v , $\xi(v) = x$ se o valor de v for x em (σ_i, σ_{i+1}) . Assim, uma configuração de sistema associada ao instante σ_{i+1} é uma tupla (X, Π, θ, ξ) , onde X é a configuração de estados maximal do estado raiz e (Π, θ, ξ) é um estímulo externo associado a σ_{i+1} .

Uma reação do sistema é um par (Υ, Π^*) , onde Υ é o conjunto de transições denominado passo e Π^* é um conjunto de eventos atômicos gerados por Υ . Uma definição informal para passo é um conjunto de transições que podem ser habilitadas e que são induzidas por estímulos externos, sendo que esse passo gera um conjunto de eventos que ocorrem como resultado das transições que o passo habilita. Todas as transições constantes em um passo Υ são disparadas simultaneamente. Um passo também pode ser definido como uma seqüência de micro-passos, que gera configurações intermediárias (micro-configurações), sendo que cada passo está contido em Υ (Francès, 2001).

Figura 3.19: Servidor de Arquivos na Especificação *Statecharts*

A Figura 3.19 exemplifica a idéia do micro-passo utilizando-se a especificação de um servidor de arquivos básico. A interpretação do modelo da figura com relação a idéia de retardos em estados é que se há transições que podem ser habilitadas, então os eventos relacionados a ela ocorrerão (Francès, 2001). O estado inicial, a primeira configuração SC1, é o conjunto de estados *default* iniciais. Admitindo que o evento *jg* (geração de *jobs*) é habilitado, a ação *inc_p* (incremento da fila do processador) é executada no estado ortogonal Proc_Q. Baseado nisso, a próxima configuração (ativar os estados básicos de cada componente ortogonal) é SC2 = (Ready, Busy.Proc_Q, Free.Proc, Busy.Disc_Q, Free.Disc, Idle.Destination). Esta configuração dispara o evento *tr[not in {Idle.Proc_Q}]* conduzindo para uma configuração intermediária (Ready, Idle.Proc_Q, Busy.Proc, Idle.Disc_Q, Idle.Disc, Idle.Destination). Esta configuração pode ser considerada como um micro-passo, que gera uma micro-configuração (uma configuração intermediária). De qualquer maneira, esta micro-configuração é refletida novamente e imediatamente para a ação *dec_p* associada com *tr[not in {Idle.Proc_Q}]* gerando SC3 = (Ready, Busy.Proc_Q, Busy.Proc, Idle.Disc_Q, Busy.Disc, Idle.Destination) (Francès

et al., 2001), (Francés, 2001).

Formalmente têm-se que (Ilari, 1987):

- **Definição 1:** uma *configuração* de estados de s é um conjunto ortogonal relativo a s cujos membros são estados básicos, definida pela tupla (X, Π, θ, ξ) .
- **Definição 2:** um *passo* é uma seqüência maximal de micro-passos definido como: seja SC uma configuração do sistema, um passo Υ a partir de SC é uma seqüência $(\mu\Upsilon_0, \dots, \mu\Upsilon_m)$ em que:
 - i. $\mu SC_0 = SC$;
 - ii. $\mu\Upsilon_0$ é um micro-passo a partir de μSC_i ; para $i=0, \dots, m$;
 - iii. μSC_{i+1} é a micro-configuração do sistema alcançada por $\mu\Upsilon_0$ para $i=0, \dots, m$;
 - iv. O conjunto $\mu\Upsilon_0 \cup \dots \mu\Upsilon_m$ é um conjunto estruturalmente consistente de transições;
 - v. Se t está habilitada numa determinada μSC_{m+1} , então $t \cup \mu\Upsilon_0 \cup \dots \mu\Upsilon_m$ não é estruturalmente maximal, ou seja, a seqüência é maximal.
- **Definição 3:** Seja SC' uma configuração do sistema e $\Upsilon = (\mu\Upsilon_0, \dots, \mu\Upsilon_m)$ um passo a partir de SC' realizado no instante de tempo σ_i . $SC = (X, \Pi, \theta, \xi)$ é a *configuração* do sistema alcançada por Υ se:
 - i. $X = \mu X_{m+1} \cup \mu Y_{m+1}$
 - ii. Π é o conjunto de eventos primitivos que ocorrem no intervalo de tempo i ;
 - iii. Para $c \in Cp$, c se mantém em σ_i se e somente se $\mu SC_{m+1} \rightarrow cr(c)$, θ é o conjunto de condições primitivas que se mantém em (σ_i, σ_{i+1}) , para algum $\sigma \geq \sigma_i$;
 - iv. Para $v \in Vp$, o valor de v em σ_i é x se e somente se $\mu\xi_{m+1}(cr(v))=x$, $\xi(v)=x$ se o valor de v em (σ_i, σ_{i+1}) é x , para algum $\sigma \geq \sigma_i$.
- **Definição 4:** Seja SC uma *configuração do sistema* no instante i e $\Upsilon = (\mu\Upsilon_0, \dots, \mu\Upsilon_m)$ um passo a partir de SC realizado em σ_i . O conjunto de *eventos* gerados por Υ é $\Pi^* = \{c \mid c \text{ gerado por } \mu\Upsilon_i \text{ para algum } i\} \cup \{c \mid \mu SC_{m+1} \rightarrow c, c \text{ atômico}\}$.
- **Definição 5:** uma *execução do sistema* é uma seqüência $\{(SC, \Upsilon_i, \Pi^*)\}_{i \geq 0}$, em que:
 - i. $SC_0 = (X_0, \Pi_0, \theta_0, \xi_0)$, X_0 é a configuração inicial de estados, e (Π_0, θ_0, ξ_0) são os estímulos externos que ocorrem no primeiro intervalo de tempo I_0 ;
 - ii. Υ_i é um passo realizado a partir de SC_i no instante de tempo σ_{i+1} , para $i \geq 0$;
 - iii. SC_{i+1} é a configuração de sistema alcançada a partir de SC_i por Υ_i , para $i \geq 0$;
 - iv. Π^* é o conjunto de eventos gerados por Υ_i , para $i \geq 0$.

3.4.7 Vantagens e desvantagens dos Statecharts

Os *Statecharts* são capazes de representar tanto recursos, quanto situações abstratas. Por intermédio de estados default pode-se representar a situação inicial do sistema e a situação corrente, por meio de passos dos *Statecharts*. Os estados tipo AND representam o paralelismo de maneira explícita.

Porém, os *Statecharts* não possuem uma representação para tratar filas e nem contemplam a individualização de clientes, não possibilitando que os clientes sejam acompanhados dentro do sistema. Todas as suas transições são tratadas como imediatas (Francês *et al.*, 2000b).

3.5 Statecharts Estocásticos

O conceito de *Statecharts* Estocásticos foi proposto na tese de doutorado de Carlos R. L. Francês (Francês, 2001), em 2001, apresentada ao Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo.

Os *Statecharts* Estocásticos baseiam-se nos *Statecharts* incluindo-se as adaptações necessárias para a obtenção de uma visão estocástica. Alguns passos, configurações e micro-configurações foram redefinidos para a obtenção de *Statecharts* Estocásticos.

Duas principais modificações foram adicionadas a especificação original: estados com tempos exponencialmente distribuídos e inclusão de probabilidades nas transições. A primeira modificação considera o *delay* como uma variável aleatória (tempo entre chegadas e tempo de serviço). Assim um estado com *delay* pode ser uma fonte que gera clientes com uma taxa de tempo exponencialmente distribuídos ou um certo servidor que concede um serviço para clientes de acordo com uma distribuição exponencial. A Figura 3.20 mostra a notação gráfica usada para diferenciar os estados com retardos e os estados imediatos. Embora a representação seja idêntica à sugerida por D. Harel (Harel, 1987) para estados com *delays* e *timeouts*, a semântica aqui é ligeiramente modificada. O sentido é que o tempo ou é a média dos tempos entre chegadas (para os casos dos estados-fonte) ou ele é um tempo médio de serviço (para os casos de servidores). Assim, o tempo que o temporizador conta, a partir da entrada no estado, é uma das variáveis aleatórias que devem ser conhecidas previamente, que possibilita a determinação da próxima configuração (Francês, 2001).

A segunda modificação refere-se à necessidade de associação de um valor probabilístico para cada possível caminho seguido por um cliente. Essa situação força o cliente a optar por um caminho entre várias possibilidades. A escolha por probabilidade é representada por um círculo com a letra *P* dentro. A Figura 3.21 apresenta a notação para a escolha por

Figura 3.20: Estados com Retardo e Imediatos (u.t.: unidades de tempo)

probabilidade.

Figura 3.21: Escolha por Probabilidade

A definição formal para a condição probabilista é:

- a) $\exists P = p_1, p_2, \dots, p_n$, onde P é o conjunto de probabilidades associadas aos eventos e p_i é cada ponto do espaço amostral definido por $P \therefore p_i \in P$, com $i = 1, 2, \dots, n$;
- b) $0 \leq p_i \leq 1, \forall i = 1, 2, \dots, n$;
- c) $\sum_{i=1}^n p_i = 1$;
- d) Assim, admite-se que $P \subset C$ $p_i \in C$.

Usa-se a notação particular $ev\{p_i\}$.

3.5.1 Redefinição de Passos e Configurações para os Statecharts Estocásticos

Em um *Statecharts* Estocástico uma configuração e um passo tem uma interpretação se todos os seus estados são considerados imediatos e outra interpretação na presença de estados com retardos associados. Assim, uma extensão da definição original é requerida para alcançar as características temporais dos *Statecharts* (Francés *et al.*, 2001).

- **Definição 1a:** Um *estado* é considerado com *retardo* quando existe uma variável τ_i , a qual quando avaliada determina:
 - i. O tempo médio entre chegadas de clientes, caso o estado seja a fonte geradora desses clientes (*Source*). Admitindo-se que os tempos entre chegadas são exponencialmente distribuídos;
 - ii. O tempo médio de serviço destinado aos clientes, caso o estado seja um servidor (em seu estado *Busy*). Admitindo-se que os tempos de serviço são exponencialmente distribuídos.
- **Definição 2a:** Um *estado* é considerado *imediato* quando seu retardo é considerado zero ($\tau_i = 0$).
- **Definição 3a:** O tempo (σ_{i+1}) da próxima configuração alcançável $SC = (X, \Pi, \theta, \xi)$ obedece às seguintes premissas:
 - i. A variável avaliada em ξ é τ_i , onde τ_i é cada retardo associado a um estado S_i cp, $i=1, \dots, n$;
 - ii. $\min(\tau_i)$ é a função que indica o menor dos tempos de retardo em uma determinada configuração SC_j , com $j=1, \dots, n$;
 - iii. O tempo gasto em cada passo é $\tau_{Y_j} = \tau_{Y_{j-1}} + \min(\tau_i)$, com $j = 1, \dots, n$. O tempo τ_{Y_j} é igual ao tempo final de uma configuração SC_i ou ao tempo inicial de uma configuração SC_{i+1} ;
 - iv. Se $j=1$, primeiro passo, então $\tau_{Y_j} = \min(\tau_i)$, pois $\tau_{Y_{j-1}}=0$;
 - v. $\tau_{i,resto} = \tau_i - \min(\tau_i)$, $\forall \tau_i \neq \min(\tau_i)$, onde $\tau_{i,resto}$ é o restante de tempo de retardo de um estado que extrapola para o próximo passo;
 - vi. $\tau_{total} = \sum_{j=1}^n \tau_{Y_j}$;
 - vii. $\forall \{\tau_i\} - \{\min(\tau_i)\}$, $\tau_i := \tau_{i,resto}$, a cada final de passo;
 - viii. Se $\tau_i = \min(\tau_i)$, então $\tau_{i,resto} = 0$ e τ_i no passo seguinte começa com valor 0.
- **Definição 4a:** Uma escolha por probabilidade leva o sistema a reações diferentes $(Y_1, \Pi_1^*), (Y_2, \Pi_2^*), \dots, (Y_n, \Pi_n^*)$, tal que $\Pi_1^* \neq \Pi_2^* \neq \dots \neq \Pi_n^*$ ou $Y_1 \neq Y_2 \neq \dots \neq Y_n$.
- **Definição 5a:** Se, em *Statechart*, as suas configurações e o tempo de seus passos são determinados pelas definições de 1a a 4a, então esse *Statechart* é Estocástico.

A Figura 3.22 utiliza o exemplo do servidor de arquivos, agora representado na notação estendida para a visão estocástica. O exemplo utiliza valores hipotéticos $\tau_{source} = 5u.l.$,

$\tau_{busy.proc} = 3u.t.$ e $\tau_{busy.disc} = 4u.t.$. Assim, é possível traçar uma linha do tempo para determinar o tempo de cada passo, que são o início e o fim de cada configuração. A ordem que as configurações ocorrem também podem ser determinadas (Francès, 2001).

Figura 3.22: Servidor de Arquivos e seus Estados com Retardos Associados

SC1 é a configuração dos estados *default* que apresenta o ingresso do sistema no estado *Source*. O estado *Source* apresenta um retardo de 5 u.t. A próxima configuração é SC2, que é função de SC1 em relação ao tempo. Se *Source* está ativo, o sistema espera 5 u.t. até gerar o primeiro cliente (evento *jg*). Depois de 5 u.t., o passo 1 é completado com a execução da ação *inc_p* (em *Proc_Q*), levando a *Busy.Proc_Q* e a *Busy.Disc_Q*. A próxima configuração, SC3, habilita de maneira imediata a transição *tr[not in(Idl.Proc_Q)]* alcançando *Busy.Proc* e *Source*, simultaneamente, por *jg*. O tempo do próximo passo e a próxima configuração serão determinados da seguinte maneira (sendo $\tau_1 = \tau_{source}$, $\tau_2 = \tau_{busy.proc}$, $\tau_3 = \tau_{busy.disc}$):

- Tempo do passo 1 é $\tau_{T1} = \min_{passo1}(\tau_{source}) = 5$, pois só há um estado com retardo (*Source*);
- $\min_{passo2}(\tau_{source}, \tau_{busy.proc}) = (5, 3) = 3$;
- Tempo do passo 2 é $\tau_{T2} = \tau_{T1} + \min(\tau_1, \tau_2) \Rightarrow \tau_{T2} = 5 + \min(5, 3) = 8$, ou seja, o próximo passo começa em 8 u.t.;
- O tempo que ultrapassa o tempo do passo 2 para os estados diferentes daqueles que possui o tempo mínimo é $\tau_{i.resto} = \tau_i - \min(\tau_1, \tau_2) = 5 - 3 = 2$, isto é, o tempo que ultrapassa do passo 2 para o passo 3 é de 2 u.t. no estado *Source*;
- Quando o passo 2 completar 8 u.t. (o seu término), então $\forall \{\tau_i\} - \{\min(\tau_i), \tau_i = \tau_{i.resto}\}$, significando que os estados que não completaram o seu retardo em um determinado

passo, começam o próximo passo com os seus tempos iguais ao restante do passo anterior. Para o exemplo, $\tau_1 = \tau_{i.resto} = 2$;

Os valores temporais para SC3 são calculados como:

- $\min_{passo2}(\tau_{source}, \tau_{busy.disc}) = (2, 4) = 2$;
- Tempo do passo 3 é $\tau_{r3} = \tau_{r2} + \min(\tau_1, \tau_3) \Rightarrow \tau_{r3} = 8 + \min(2, 4) = 10$, ou seja, o próximo passo começa em 10 u.t

O tempo que ultrapassa o tempo do passo 3 para os estados diferentes daquele que possui o tempo mínimo é $\tau_{3.resto} = \tau_3 - \min(\tau_1, \tau_3) = 4 - 2 = 2$. Portanto, o tempo que ultrapassa do passo 3 para o passo 4 é de 2 u.t. no estado *Busy.Disc*.

Na próxima seção serão apresentados alguns templates para prover a representação de características (estados e eventos padrões) de sistemas de filas .

3.5.2 Templates e Eventos-Padrão para Sistemas de Filas

Em (Francès, 2001) são descritos quatro *templates* e seus eventos-padrão para especificar um sistema de filas. Em *Statecharts* Estocásticos *template* é um conjunto de estados (possivelmente unitário) que define o funcionamento básico de um determinado componente do sistema .

Para fontes geradoras de clientes há dois tipos de templates: para estados com retardos e estados imediatos. No primeiro tipo de fonte geradora (*template* (a)), não há a geração imediata do primeiro cliente, pois o estado *default* é um estado com retardo, isto é, uma vez alcançado o estado, deve-se esperar o término do τ_i associado a ele (Figura 3.23(a)).

No segundo tipo de fonte geradora (*template* (b)), há a geração de cliente no instante zero, uma vez que o estado *default* é um estado imediato (Figura 3.23(b)).

Figura 3.23: *Templates* para Fontes Geradoras de Clientes

A ação $inc/-s$ acrescenta o cliente gerado na fila de um servidor s e o evento ig é responsável pela geração de um cliente.

O *template* que representa a fila (Figura 3.24) possui dois subestados: *Idle* e *Busy*. A fila, por definição, se encontra vazia (*Idle*) e a cada geração de cliente, uma unidade é acrescentada na fila e nesse momento o estado da fila passa para *Busy*. A ausência de clientes na fila muda o estado para *Idle*.

Figura 3.24: *Template* para Fila de Servidores

O *template* servidor contém dois subestados: *Idle* e *Busy*. No subestado *Busy* há um retardo de τ_i , que significa um valor médio de uma distribuição de probabilidade. A Figura 3.25 apresenta a representação gráfica para o *template* servidor.

Figura 3.25: *Template* para Servidor

Quando há clientes na fila, o evento $tr[not\ in\ (Idle.Server_i.Q)]$ pode ser executado, se o estado *Idle* estiver ativo, sua transição é habilitada e a fila decrementada. Ao término do atendimento em *Busy*, o evento *eos* pode ser executado por meio de uma de suas condições probabilísticas. O cliente pode ir a outro servidor, indo para a respectiva fila (inc_S_{i+1}), ou abandonar o sistema.

O *template destination* (sorvedouro) é formado por dois subestados: *Idle* e *Out_System*. O subestado *Idle* indica que o *destination* está a espera de alguma resposta. O subestado *Out_System* significa que ele já recebeu a resposta, podendo então deixar o sistema. A Figura 3.26 representa notação gráfica para o *template* sorvedouro.

Figura 3.26: *Template* para Sorvedouro

Francês, em (Francês, 2001), padronizou os eventos e nomes de estados da especificação baseada nos *templates*. O que não impede que novos eventos sejam criados para aumentar a clareza da especificação. A Tabela 3.1 apresenta o rótulo dos eventos e a semântica atribuída a cada um deles.

Tabela 3.1: Eventos-Padrão e suas Semânticas

Rótulo do Evento	Semântica Atribuída
jg (<i>Job Generation</i>)	Geração de um cliente obedecendo a uma determinada distribuição de probabilidade.
inc_S (<i>Increment of Server Queue</i>)	Incrementa a fila de um determinado servidor S.
dec_S (<i>Decrease of Server Queue</i>)	Decrementa a fila de um determinado servidor S.
tr[not in (Idle.Server_Q)]	Assegura que a fila do servidor não está vazia, e que o próximo cliente pode ir ao servidor.
cos{p} (<i>End of Service</i>)	Indica o término do atendimento a um cliente, e uma escolha probabilística para determinar o caminho a ser seguido pelo cliente.
exit	Indica que o cliente sai do sistema que o provê de um determinado serviço e vai até um sorvedouro.

A Tabela 3.2 apresenta um conjunto de nomes de estados possíveis.

3.6 *Queuing Statecharts*

Francês, em sua tese de doutorado (Francês, 2001), propôs os *Queuing Statecharts*. *Queuing Statecharts* é uma especificação que une a capacidade das redes de filas em representar o caminho linear que um determinado cliente traça durante a sua passagem pelo sistema e a habilidade dos *Statecharts* em representar adequadamente a complexidade que os servidores necessitam para que as suas descrições se tornem esclarecedoras (Francês *et al.*, 2002a), (Francês *et al.*, 2002b), (Francês, 2001), (Francês *et al.*, 2001).

A próxima seção provê os meios necessários para adaptar os *Statecharts* para implementar as características descritas.

Tabela 3.2: Nomes de Estados-Padrão e suas Semânticas

Nome de Estados	Semântica Atribuída
Source	Estado que representa um componente genérico do sistema responsável pela geração de um cliente, obedecendo a uma determinada distribuição de probabilidade.
Server-Q	Estado que representa a fila de um determinado servidor. Apesar da fila ser representada por apenas um lugar para armazenamento, admite-se a possibilidade da fila infinita, por conveniência da fundamentação matemática.
Server	Estado que representa o componente responsável pela prestação de um determinado serviço do sistema. Assim como acontece em <i>Source</i> , o serviço prestado obedece a uma distribuição de probabilidade.
Destination	Estado que representa o componente do sistema que absorve os clientes após a execução de um serviço.

3.6.1 Definição de Queuing Statecharts

Os *Queuing Statecharts* (QS) são definidos através de seus componentes conforme a seguir (Frances, 2001):

Centro de Serviço (Service Center):

- **Definição 1b:** o estado s' é um servidor se e somente se:
 - i. $\exists S^s \subset S$, que é o conjunto de todos os servidores do QS, tal que todo $s' \in S^s$;
 - ii. $s' \in S \wedge \rho(s') = 2 = \{Idle.Server, Busy.Server\}$;
 - iii. $\psi(s') = OR$;
 - iv. $\delta(s') = \{Idle.Server\}$;
 - v. As ações de incremento e decremento (inc-, dec) são sempre realizadas por um servidor s' em uma fila s'' .
- **Definição 2b:** o estado s'' é uma fila, se e somente se:
 - i. $\exists S^q \subset S$, que é o conjunto de todas as filas do QS, tal que todo $s'' \in S^q$;
 - ii. $s'' \in S \wedge \rho(s'') = \emptyset \therefore s''$ é um estado básico;
 - iii. s'' possui uma variável Q_j (clientes na fila), com $j = 1, 2, \dots, k$, que representa a variável aleatória número de clientes na fila;

iv. Q_j é definida como:

(a) $Q_j \in V$, op é uma operação algébrica, então $op(Q_j) \in V$;

(b) $Q_j, v \in V, R \in \{=, >, <, \neq, \geq, \leq\}$, então $v R Q_j \in C$;

As únicas operações realizadas em Q_j são: $Q_j := Q_j + w, Q_j := Q_j - w$, com $w = 1, 2, \dots, n$;

• **Definição 3b:** o estado s é um centro de serviço, se e somente se:

i. $s \in S \wedge \rho(s) = 2 = \{\text{Server-Q}, \text{Server}\}$, onde $(S^s \cup S^q) \subset S$;

ii. $\psi(s') = \text{AND} \therefore \text{Server-Q} \perp \text{Server}$;

Ou seja, $\rho(s) \neq \emptyset$ e $\psi(s') = \text{AND}$, então trata-se de uma decomposição AND de s ;

• **Definição 4b:** s^o é um estado fonte (*Source*) se e somente se:

i. $s^o \in S \wedge \rho(s^o) = 1 = \{\text{Ready}\}$;

ii. $\text{Ready} (\in s^o)$ possui uma variável τ_i , que representa a variável aleatória tempo entre geração de clientes, obedecendo a uma distribuição de probabilidade exponencial;

• **Definição 5b:** os é um estado fonte sorvedouro (*Destination*) se e somente se:

i. $^os \in S \wedge \rho(^os) = \emptyset$ e os é $\therefore$ um estado básico.

3.6.2 Templates e Eventos-Padrão para Queuing Statecharts

São estabelecidos *templates* para cada um dos componentes fonte (Figura 3.27), fila (Figura 3.28), servidor (Figura 3.29), centro de serviço (Figura 3.30) e sorvedouro (Figura 3.31).

Figura 3.27: *Template* para Fonte (*Source*)

A especificação *Queuing Statechart* manteve todas as definições apresentadas em (Harel, 1987) e as especificações definidas em (Francès, 2001) para os *Statecharts* Estocásticos.

Os *Queuing Statecharts* possuem um conjunto de eventos-padrão que definem o comportamento básico de um sistema de filas. A Tabela 3.3 apresenta o conjunto de eventos definidos para *Queuing Statecharts*.

Figura 3.28: *Template* para Fila (*Queue*) Figura 3.29: *Template* para Servidor (*Server*) Figura 3.30: *Template* para Centro de Serviço (*Service Center*) Figura 3.31: *Template* para Sorvedouro (*Destination*)

3.7 Considerações Finais

Este capítulo abordou as três técnicas de representação de modelos mais referenciadas na literatura (redes de fila, redes de Petri e *Statecharts*) e duas novas abordagens (*Statccharts* Estocásticos e *Queuing Statecharts*).

Redes de fila é empregada para representar sistemas em que existam apenas os elementos fila e servidor. Contudo, não representam de forma mais abrangente os estados e as transições em que o sistema é submetido no decorrer do tempo. Embora as redes de Petri e *Statecharts* consigam representar o sistema por meio dos seus estados e suas transições, elas possuem certas limitações. Por exemplo, as redes de Petri não representam adequadamente modelos paralelos e os *Statecharts* não conseguem representar o caminho linear seguido por um determinado cliente no sistema.

Os *Statecharts* Estocásticos e *Queuing Statecharts* são uma nova alternativa para

Tabela 3.3: Eventos-Padrão de *Queuing Statecharts* e suas Semânticas

Rótulo do Evento	Semântica Atribuída
yg (<i>Job Generation</i>)	Geração de um cliente obedecendo a uma determinada distribuição de probabilidade.
inc $_Q$ (<i>Increment the number of clients in the Server Queue</i>)	Incrementa a variável aleatória que representa o número de clientes na fila de um determinado servidor S_j .
dec $_n$ (<i>Decrease the number of clients in the Server Queue</i>)	Decrementa a variável aleatória que representa o número de clientes na fila de um determinado servidor S_j .
tr($Q_j \geq 1$)	Assegura que a fila do servidor não está vazia, e que o próximo cliente pode ir ao servidor.
cos{p} (<i>End of Service</i>)	Indica o término do atendimento a um cliente, e uma escolha probabilística para determinar o caminho a ser seguido pelo cliente.
exit	Indica que o cliente sai do sistema que o provê de um determinado serviço e vai até um sorvedouro.

especificar sistemas baseadas em *Statecharts*. Cada uma das especificações acima possuem um conjunto de *templates* que representam os componentes básicos de um sistema de filas. As duas extensões permitem a representação de quaisquer componentes do sistema e possuem as características de *Statecharts* como: paralelismo, hierarquia, mecanismos de comunicação, etc. Porém, *Statecharts* Estocásticos não conseguem representar o caminho seguido pelos clientes no sistema, o que é suavizado em *Queuing Statecharts*.

As principais características da representação gráfica de cada técnica é sintetizada na Tabela 3.4 (Francès *et al.*, 2001).

Tabela 3.4: Características das Técnicas de Especificação

Técnica	Redes de Fila	Redes de Pe- tri	Statecharts	Statecharts Estocásticos	Queuing Statecharts
Característica					
Paralelismo Explícito de Componentes	Especificado subjeticamente	Especificado subjeticamente	Especificado explicitamente	Especificado explicitamente	Especificado explicitamente
Mecanismos de Comunicação entre Componentes	Não especificado	Especificado através dos <i>tokens</i>	Especificado através dos eventos, ações e condições	Especificado através dos eventos, ações e condições	Especificação reforçada pelas setas que unem os componentes do modelo
Hierarquia entre Componentes	Não especificado	Especificado, na Notação Original	Especificado através de eventos, ações e condições	Especificado através de eventos, ações e condições	Especificado através de eventos, ações e condições
Representação do Ca- minho Linear Seguido por um Cliente	Especificado pelas setas que unem os centros de serviço	Especificado através do fluir dos <i>to- kens</i> dentro do sistema	Especificado subjeticamente através dos eventos e ações	Especificado subjeticamente através dos eventos e ações	Especificado explicitamente
Marcação Inicial	Não especificado	Especificado através dos <i>tokens</i>	Especificado através dos estados <i>default</i>	Especificado através dos estados <i>default</i>	Especificado através dos estados <i>default</i>
Individualização de Clientes	Não especificado	Especificado através dos <i>tokens</i>	Não especificado	Não especificado	Não especificado

ASDA - Um Ambiente de Simulação Distribuída Automático

4.1 Considerações Iniciais

Como visto no Capítulo 2, a simulação é uma das ferramentas para a avaliação de desempenho de sistemas computacionais (Pegden *et al.*, 1995) e sua utilização requer que algumas etapas sejam consideradas. Primeiramente, é necessário especificar o modelo, abstraindo as características mais importantes do sistema. A especificação do modelo é feita utilizando-se uma das técnicas para representação de modelos descritas nos Capítulos 2 e 3. Tendo o sistema sido modelado, é necessário transformar o modelo em um programa de simulação.

A simulação pode ser aplicada em diferentes áreas, devido, principalmente, a sua flexibilidade e baixo custo. Esta técnica permite que quaisquer alteração no sistema seja refletida com facilidade no programa de simulação. Apesar de todas as vantagens apresentadas pela simulação para avaliação de sistemas computacionais, a simulação seqüencial apresenta algumas desvantagens (por exemplo, o tempo da simulação) quando é preciso simular modelos complexos com grande quantidade de detalhes (M. *et al.*, 2000).

Para resolver este problema, utiliza-se simulação distribuída. A simulação distribuída oferece algumas vantagens sobre a simulação seqüencial: redução do tempo de execução da simulação, execução da simulação em um conjunto de computadores distribuídos geograficamente, integração de computadores que executam em máquinas de diferentes fabricantes e tolerância a falhas (Fujimoto, 1999), (Fujimoto, 2001).

Um dos objetivos dessa dissertação de mestrado é avaliar duas técnicas para repre-

sentação de modelos. Estas técnicas deverão ser avaliadas considerando diversos aspectos. Uma vez selecionada uma das técnicas, esta deverá ser aplicada no ambiente de simulação distribuído automático - ASDA, em desenvolvimento no grupo de Sistemas Distribuídos e Programação Concorrente do Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo. Desta forma, este capítulo apresenta uma breve descrição sobre simulação distribuída e tem como objetivo descrever as características gerais do ASDA e detalhar as partes deste ambiente que estão diretamente relacionadas com este projeto. Isto é, serão descritos com maior enfoque os módulos relacionados com o desenvolvimento do modelo a ser simulado e a interface com o usuário.

4.2 Simulação Distribuída

A simulação distribuída tem sido desenvolvida com o objetivo de diminuir o tempo da simulação sequencial. Dessa maneira, duas soluções tem sido propostas: SRIP (*Single Replication in Parallel*) e MRIP (*Multiple Replications in Parallel*). Estas soluções diferem, basicamente, no modo como o programa de simulação é dividido em processos lógicos e como, esses processos são mapeados nos processadores disponíveis.

A técnica SRIP é baseada na decomposição do modelo de simulação em processos lógicos, executando em diferentes processadores e a comunicação entre eles é feita por meio de protocolos de passagem de mensagem. Um problema crítico na técnica SRIP é a garantia do sincronismo entre os vários processos que compõem o programa de simulação. Vários protocolos tem sido desenvolvidos e agrupados dentro de duas categorias: protocolos conservativos e protocolos otimistas (Fujimoto, 1999).

A principal característica dos protocolos conservativos é a execução de um evento somente quando é segura a sua execução, ou seja, o evento só é executado quando há garantias de que nenhum outro evento com *timestamp* menor chegará para ser executado. Enquanto isto não é garantido, o processo fica bloqueado o que pode gerar uma queda no desempenho e a situação de *deadlock* (Fujimoto, 1999).

Os mecanismos conservativos diferem no modo como o deadlock é tratado. Alguns deles previnem a ocorrência de deadlock e outros deixam que o deadlock ocorra e depois utilizam algum mecanismo para recuperar o sistema.

O protocolo conservativo CMB (Chandy, Misra e Bryant) (Bryant, 1977), (Chandy & Misra, 1979) previne *deadlock* por meio da utilização de mensagens nulas. A mensagem nula é utilizada somente para efeitos de sincronização e não corresponde a nenhuma atividade real do sistema.

Os protocolos otimistas não impedem os erros de causa e efeito, assim não é preciso determinar quando é seguro processar um evento. Quando um erro é detectado um procedimento é chamado para recuperar o sistema e voltar a um estado seguro (*rollback*). O mecanismo *Time Warp* é o mais conhecido protocolo otimista e se baseia no paradigma de *Virtual Time* (Jefferson, 1985). O mecanismo *Time Warp* permite que um processo lógico execute eventos em seu tempo de simulação local, denominado *Local Virtual Time (LVT)*, assim que exista uma mensagem na sua lista de entrada (Ayani, 1998).

Na técnica MRIP várias instâncias de um mesmo programa de simulação seqüencial são executadas independentemente em diferentes processadores. Cada instância envia os resultados para um analisador global, que avalia o resultado final. Uma das vantagens da utilização da técnica MRIP é que ela pode ser aplicada em qualquer programa de simulação.

4.3 ASDA

O ASDA foi proposto por S. M. Bruschi (Bruschi, 2002), em sua tese de doutorado apresentada ao Instituto de Ciências Matemáticas e de Computação na Universidade de São Paulo. O ASDA é um ambiente de simulação voltado à avaliação de desempenho de sistemas computacionais.

O objetivo do ASDA é oferecer ao usuário um ambiente automático para o desenvolvimento de simulação, seguindo as características descritas nos ambientes automáticos, que são: interface amigável, geração e compilação automática do código de simulação, execução e análise dos resultados. Os usuários de um ambiente desse tipo apresentam diferentes tipos de conhecimento, como por exemplo (Bruschi, 2002):

- Grande conhecimento em simulação, mas pouco conhecimento em computação paralela;
- Conhecimento superficial tanto de simulação como de computação paralela;
- Conhecimento em simulação e computação paralela, mas que disponham de pouco tempo para o desenvolvimento da simulação;
- Usuários que desejam simular o mesmo sistema de maneira seqüencial e distribuída de forma a analisar o desempenho, etc.

De forma a atender a diferentes perfis de usuários, o ASDA deve atender a alguns requisitos:

1. Oferecer ao usuário um ambiente de fácil aprendizagem e utilização;

2. Possibilitar a geração completa de um problema de simulação para um usuário menos experiente;
3. Oferecer flexibilidade necessária para que um usuário mais experiente possa modificar os programas gerados;
4. Possibilitar a utilização de simulações seqüências já desenvolvidas;
5. Oferecer diretrizes para que o usuário possa escolher entre as abordagens para simulação distribuída. Se o usuário preferir, o ambiente deve ser capaz de selecionar a abordagem mais adequada;
6. Facilitar a obtenção de dados confiáveis através da avaliação estatística dos resultados gerados pela simulação;
7. Minimizar o tempo de execução da simulação, oferecendo um escalonamento de processos eficiente.

Dessa forma, o ASDA foi estruturado em um diagrama modular (Figura 4.1) composto por sete módulos (Bruschi, 2002), sendo:

- **Módulo de Interface com o Usuário:** oferece ao usuário uma interface que possibilita um acesso fácil às opções disponibilizadas pelo ambiente. Ao mesmo tempo oferece flexibilidade para usuários que prefiram tomar suas próprias decisões. É responsável ainda por oferecer ao usuário um editor gráfico onde será definido o modelo a ser simulado;
- **Módulo Avaliador:** baseando-se nas variáveis do ambiente e no modelo a ser simulado, esse módulo orienta o usuário na difícil tarefa de escolher a melhor abordagem para sua simulação. Nesse módulo deve estar concentrado todo o conhecimento necessário para decidir ou para orientar o usuário sobre a melhor abordagem para a simulação. Este módulo é dividido em 3 níveis: nível 1, responsável pela verificação da consistência do modelo representado; nível 2: responsável por auxiliar na escolha de utilização das abordagens MRIP (*Multiple Replication in Parallel*) ou SRIP (*Single Replication in Parallel*); nível 3: responsável por auxiliar na escolha de utilização dos protocolos de sincronização conservativos, otimistas ou conservativos que migrem para otimistas;
- **Módulo Gerador:** baseando-se na especificação gráfica e na decisão do módulo avaliador, gera o código do programa de simulação;

Figura 4.1: Diagrama Modular do ASDA

- **Módulo Executor:** responsável por controlar a execução dos programas de simulação;
- **Módulo de Interface de Software:** responsável por fornecer uma interface entre os programas de simulação já desenvolvidos e o módulo replicador;
- **Módulo Replicador:** gerencia as replicações da simulação e garante a obtenção de resultados confiáveis através da abordagem MRIP;
- **Módulo Escalonamento:** responsável pelo escalonamento dos processos gerados pelo módulo executor.

4.3.1 Módulo de Interface com o Usuário

A finalidade do módulo de interface com o usuário é facilitar a interação do usuário com o ASDA. Um sistema tutor auxilia o usuário dando informações sobre a modelagem do sistema e a avaliação de desempenho.

O módulo de interface realiza as seguintes funções (Bruschi, 2002):

1. Oferece ao usuário uma interface amigável;
2. Identifica e gerencia os usuários do sistema;
3. Recebe os dados definidos pelo usuário e os armazena em bases de dados;
4. Disponibiliza os dados armazenados para os usuários e para os outros módulos do sistema;
5. Permite a especificação gráfica do modelo a ser simulado

Com o objetivo de se obter uma interface amigável no ASDA utiliza-se conceitos de janelas, menus, caixas de diálogo, barras de ferramentas e barras de rolagem. Na tela inicial o usuário realiza a identificação. Após essa identificação é realizado a identificação do tipo de usuário. Há dois níveis de usuários (Bruschi, 2002):

- Super-usuário: determina os valores padrões das variáveis do ambiente e cadastra novos usuários. Ao super-usuário não é permitido realizar simulações;
- Usuário de simulação: define os modelos e executa as simulações.

A interação entre os dados fornecidos pelos usuários ou por outros módulos do ASDA para serem armazenados na base de dados e solicitados pelos usuários já armazenados na base de dados são de responsabilidades do módulo de interface com o usuário. A base de dados referentes às informações dos usuários do ASDA é composta por:

- Nome do usuário, para fazer login no sistema;
- Senha, para validar o usuário;
- Configuração padrão do usuário: características padrões de cada usuário que são lidas tão logo o usuário entre no sistema;
 - Técnica de modelagem;
 - Linguagem utilizada;
 - Abordagem;
 - Tamanho do warm-up;
 - Tamanho do lote;

– Plataforma;

- Lista de modelos, que possui as informações sobre os modelos desenvolvidos pelo usuário;

Enquanto o usuário está definindo o seu modelo as seguintes informações vão sendo atualizadas na lista de modelos:

- Nome do arquivo de especificação gráfica;
- Nome do arquivo de programa de simulação;
- Abordagem utilizada na simulação (Sequencial, MRIP, SRIP ou SRIP + MRIP);
- Linguagem/extensão funcional utilizada;
- Plataforma (topologia da rede + arquitetura computacional) utilizada na simulação;
- Número de replicações;
- Método de análise;
- Resultados obtidos;
- Nível de confiança;
- Precisão;
- Data de criação;
- Data da última modificação.

A especificação gráfica contém uma janela para especificação do modelo e as barras de ferramentas com os recursos disponíveis para a criação do modelo. Os recursos irão depender do tipo de abordagem escolhida.

Enquanto o usuário define o modelo, a verificação da consistência é realizada pelo módulo Avaliador, nível 1. O modelo especificado pelo usuário é armazenado na forma de um grafo. Cada centro de serviço definido no modelo recebe um número e é representado por um nó no grafo e as ligações entre os centros de serviços são representadas pelos arcos. Se existir probabilidades, estas são representadas como pesos dos arcos. Para cada tipo de nó armazenado são armazenadas diferentes tipos de informações, conforme detalhado a seguir:

- **Nó Centro de Serviço**

- *Identificador*: cada centro de serviço deve ser identificado por um número que será definido pelo sistema;
- *Nome do centro de serviço*: nome do centro de serviço;
- *Número de filas*: número de filas que cada centro possui;
- *Número de servidores*: número de servidores que cada centro possui;
- *Distribuição de serviço*: distribuição de probabilidade do tempo para que um servidor execute um serviço;
- *Parâmetro a ser analisado*: determina qual (quais) parâmetro(s) do centro de serviço vai(vão) ser analisados.

- **Nó Saída**

- *Identificador*: identificador do nó saída.

- **Nó Entrada**

- *Identificador*: identificador do nó entrada;
- *Distribuição de chegada*: distribuição de probabilidade do tempo entre as chegadas dos clientes.

- **Nó Posse Múltipla:**

- *Identificador*: identificador de posse múltipla;
- *Nome do recurso passivo*: nome do recurso passivo que irá ser representado;
- *Tipo do nó posse múltipla*: *alocação* caso o nó esteja representando uma alocação e *liberação* caso o nó esteja representando uma liberação.

- **Nó Servidor sem fila:**

- *Identificador*: identificador do nó servidor sem fila;
- *Nome do servidor*: nome do servidor sem fila;
- *Política de reescalonamento*: política a ser seguida para uma nova tentativa de utilização do recurso caso ele esteja ocupado;
- *Distribuição de serviço*: distribuição de probabilidade do tempo para que o servidor execute um serviço.

A base de dados do usuário contém informações de cada usuário e os modelos definidos por ele, e a base de dados referente às informações do sistema contém as informações da configuração do ambiente e informações referentes aos módulos Avaliador, Executor, Replicador e Escalonamento.

As informações sobre o modelo que serão armazenadas são:

- Nome do modelo: nome que o modelo será referenciado no programa de simulação;
- Tempo de execução: tempo que a simulação será executada caso seja uma simulação terminante;
- Número de ciclos: número de ciclos que os clientes devem fazer caso seja uma simulação terminante em modelos fechados;
- Tamanho do warm-up: número de observações da simulação que devem ser descartadas na análise estatística, caso não seja utilizado o módulo Replicador;
- Tamanho do lote (batch): número de observações da simulação contidas em cada lote para análise dos métodos que utilizam lotes (Média dos lotes e Padronização de Séries Temporais).

As informações referentes à configuração do ambiente são:

- *Path* (caminho): local onde os arquivos do ambiente são buscados;
- *Criar arquivos de backup*: se devem ser criados arquivo de *backup* ou não;
- *Salvar mudanças ao sair*: se as mudanças efetuadas no ambiente devem ser salvas ou não;
- *Tamanho da lista de arquivos recente*: número de arquivos que foram recentemente utilizados pelo usuário;
- *Abrir último projeto ao entrar*: se deve ser aberto o último projeto quando o sistema é iniciado.

As informações referentes ao módulo Avaliador são:

- Tabela de decisão: tabela que possui como campos os fatores que decidem quando as abordagens MRIP ou SRIP devem ser utilizadas;
- Método de busca: método pelo qual será feita a busca na tabela de decisão.

As informações referentes ao módulo Executor são:

- Arquitetura: arquitetura sobre a qual a simulação será executada;

- Máquinas: conjunto de máquinas que farão parte da máquina virtual onde a simulação será executada.

As informações referentes ao módulo Replicador são:

- Método padrão para análise das observações, o qual pode ser: Média dos lotes, Análise espectral ou padronização de séries temporais;
- Tamanho padrão do período transiente;
- Tamanho padrão do tamanho do lote;
- Valor da variável k para detecção do período transiente;
- Nível de confiança para o término da simulação;
- Precisão do valor a ser analisado na simulação.

As informações referentes ao módulo Escalonamento são:

- Política a ser adotada no escalonamento;
- Aplicação que será executada em cada máquina para determinar a potência computacional.

4.3.2 Módulo Avaliador

O módulo Avaliador auxilia o usuário verificando as informações fornecidas e também auxiliando nas várias decisões que devem ser tomadas no sistema. O módulo avaliador possui três níveis:

- Nível 1: O nível 1 do módulo Avaliador atua em conjunto com o módulo de Especificação Gráfica. Ele é responsável pela verificação da consistência do modelo (nível 1.1) e se todos os parâmetros foram fornecidos, de modo que o usuário forneça um modelo que possa ser codificado sem erros (nível 1.2). Nos casos em que os valores para os parâmetros podem levar a resultados incoerentes na simulação ele emite um aviso ao usuário.
- Nível 2: auxilia o usuário na escolha entre simulação seqüencial ou distribuída SRIP ou MRIP;
- Nível 3: oferece os subsídios para a decisão entre os protocolos de sincronização SRIP.

4.3.3 Módulo Gerador

O módulo Gerador é responsável pela geração do programa de simulação baseado na especificação (modelo + parametrização) do sistema fornecido pelo usuário.

O ASDA deve permitir a geração de código em algumas linguagens/extensões funcionais pré-determinadas. Para cada linguagem/extensão deve haver dois arquivos, conforme representado na Figura 4.2:

Figura 4.2: Estrutura de Arquivos no Módulo Gerador

- um arquivo denominado gabarito.dat, que são trechos de código na linguagem/biblioteca alvo combinados com comandos que indicam onde as informações necessárias são encontradas;
- um programa gerador que, a partir dos dados da especificação e do arquivo gabarito.dat, gera o programa de simulação.

4.4 Considerações Finais

Neste capítulo foi apresentado o ASDA (Ambiente de Simulação Distribuída Automático), que é um ambiente de desenvolvimento de simulação, que pode ser usado por usuários com diferentes tipos de experiência e que queiram minimizar o tempo necessário para o desenvolvimento de programas de simulação.

O ASDA oferece ao usuário um ambiente flexível e amigável para o desenvolvimento, visto que oferece a possibilidade de representar o modelo, parametrizá-lo e definir todas as características desejadas para executar um programa de simulação por intermédio de uma

interface gráfica de fácil interação. Este novo ambiente para a execução de simulação tem um potencial de auxiliar no desenvolvimento e uso da simulação distribuída, oferecendo uma abordagem fácil e automática.

Implementação do Módulo de Interface com o Usuário

5.1 *Considerações Iniciais*

O módulo de interface com o usuário tem como objetivo facilitar a interação entre o usuário e o ASDA. Ele disponibiliza ao usuário uma interface que facilita o acesso as opções oferecidas, um ambiente de fácil utilização e aprendizado e flexibilidade para a modificação dos modelos gerados.

A interface com o usuário oferece um editor gráfico em que o modelo a ser estudado e seus parâmetros são introduzidos no ASDA utilizando a técnica de especificação gráfica *Queuing Statecharts*. A escolha da técnica de especificação gráfica é justificada na seção 5.2.

O módulo de interface foi desenvolvido na linguagem Java (Gosling & McGilton, 1996) e utilizando o sistema gerenciador de banco de dados MySQL (Suehring, 2002). A escolha dessas ferramentas baseia-se em suas características que oferecem um software de boa qualidade, de arquitetura neutra e portátil. Algumas considerações a respeito da escolha destas ferramentas são apresentadas nas seções 5.3 e seção 5.4.

Algumas considerações a respeito da escolha da ferramenta para a implementação do módulo de interface com o usuário e a base de dados do ASDA visando a obtenção de um software de boa qualidade, de arquitetura neutra e portátil são apresentados na seção 5.3 e seção 5.4.

A seção 5.5 descreve a estrutura de dados utilizada para armazenar o modelo especificado pelo usuário e as classes que compõem essa estrutura. Os principais aspectos relacionados a implementação da interface gráfica do ASDA e da verificação de consistência

do modelo (módulo avaliador nível 1) são descritos na seção 5.6.

5.2 A Escolha da Técnica de Especificação Gráfica

Grande parte dos sistemas de computação que podem ser simulados apresentam a característica de possuir servidores e filas. Como pode ser observado no capítulo 3 as abordagens para representação de modelos *Statecharts* Estocásticos e *Queuing Statecharts* apresentam uma nova alternativa para a especificar estes sistemas. Ambas as técnicas possuem a capacidade de detalhar a complexidade que os servidores necessitam para que suas descrições se tornem esclarecedoras, permitem que quaisquer componentes do sistema sejam representados e ainda comportam a representação de características como paralelismo, mecanismos de comunicação e hierarquia.

Para a representação de modelos no ASDA optou-se pela especificação *Queuing Statecharts* uma vez que ela possui a capacidade de representar o caminho linear que o cliente traça durante a sua passagem pelo sistema, característica ausente em *Statecharts* Estocásticos, mostrando-se assim uma especificação mais completa para sistema de filas.

5.3 A Escolha da Ferramenta para a Implementação

A linguagem escolhida para a implementação do Módulo de Interface com o Usuário e do Módulo Gerador foi Java. A escolha baseou-se na avaliação das características dessa linguagem.

Primeiramente, Java é uma linguagem de alto nível, muito parecida com C/C++, porém alguns recursos foram removidos, principalmente aqueles que propiciam más práticas de programação, ou os que são raramente usados. Um exemplo disso é que Java não possui sobrecarga de operadores, ponteiros e sua aritmética, *structs*, *unions*, herança múltipla, arquivos .h, diretivas de pré-processamento e a memória alocada dinamicamente é gerenciada pela própria linguagem.

Java é uma linguagem de programação orientada a objetos. Do ponto de vista do programador, isso significa que o programa é construído em função dos dados a serem manipulados e dos métodos que manipulam esses dados. O código é organizado em classes, que podem estabelecer relacionamentos de herança simples entre si. Somente a herança simples é permitida em Java.

Outra característica atrativa de Java é o seu processamento distribuído. As chamadas a funções de acesso remoto (*sockets*) e os protocolos Internet mais comuns (HTTP, FTP,

Telnet, etc.) são suportadas em Java, de forma que a elaboração de aplicativos baseados em arquiteturas cliente-servidor é facilmente obtida. Um programa Java é capaz de utilizar recursos distribuídos da rede para crescer dinamicamente, adaptando-se às necessidades do usuário.

Java tem o suporte a multitarefa embutido na linguagem: um programa Java pode possuir mais de uma linha de execução (*thread*). Por exemplo, cálculos extensos, que em geral demandam muito tempo do processador, podem ser escritos em uma *thread*, e a parte de interface com o usuário, que depende mais dos periféricos de I/O que do processador, pode ser executada em outra *thread*. Programação concorrente em geral é uma tarefa difícil, mas Java fornece diversos recursos de sincronização de processos que tornam a programação mais simples.

Programas escritos em Java executam uma verificação em tempo de execução quanto aos acessos de memória, abertura de arquivos e uma série de eventos que podem gerar a interrupção da execução do programa, mas que em Java geram exceções. Por exemplo, em C++ a simples tentativa de abertura de um arquivo inexistente pode obrigar ao programador reiniciar o computador, o mesmo não ocorre com programas Java.

Em Java não há a necessidade de se preocupar com o gerenciamento de memória como em C++. Em C++, todo bloco de memória alocado dinamicamente (com **new**, **malloc** ou função similar) deveria ser liberado quando não fosse mais usado (com **free**, **delete** e parentes próximos). Isso acarretava diversos problemas mesmo ao programador mais experiente, que tinha que manter sempre um controle das áreas de memória alocadas para poder liberá-las em seguida. Java, ao contrário, utiliza-se de um conceito que é o de *garbage collection* (coleta de lixo). Sua função é a de varrer a memória de tempos em tempos, liberando automaticamente os blocos que não estão sendo utilizados. Se por um lado isso pode deixar o aplicativo um pouco mais lento, por manter uma *thread* paralela que dura todo o tempo de execução do programa, evita problemas como referências perdidas e avisos de falta de memória quando sabe-se que há espaço disponível na memória.

Além dessas características, a linguagem Java foi desenvolvida para suportar aplicações desenvolvidas em ambientes heterogêneos. Isto é, as aplicações são capazes de executar em uma variedade de arquiteturas de hardware e dentro desta variedade de plataformas de hardware as aplicações devem executar em cima de uma variedade de sistemas operacionais e operar em conjunto com múltiplas linguagens de programação (Gosling & McGilton, 1996).

A ferramenta IDE (*Integrated Development Environments* - Ambiente de Desenvolvimento Integrado) utilizada para o desenvolvimento do ASDA foi a plataforma Eclipse.

O Eclipse é um *framework* para integração de diferentes tipos de aplicação. Uma

dessas aplicações é o Java IDE-JDT (Java *Development Tooling* - Ferramenta de Desenvolvimento Java) que já vem com a plataforma Eclipse.

O Eclipse foi escolhido por concentrar todas as funcionalidades para desenvolver aplicações Java, além de ser uma IDE livre e de código aberto. O Eclipse contém sinalizadores para indicar erros no código e sugere soluções para erros comuns. Ao iniciar a digitação de um nome de um método ele oferece uma lista com possíveis métodos correspondentes. Ao usar o recurso para renomear um arquivo todas as referências àquele arquivo são renomeadas automaticamente. Possui formatador de código e todo o código escrito e comentado utilizando as *tags* Javadoc (ferramenta para geração de documentação API no formato HTML para comentários no código fonte) geram uma API com toda a documentação da aplicação.

Na construção da interface gráfica e do gerador de programas de simulação foram utilizados Java 2 *Platform, Standard Edition*, versão 1.4.2 (J2SE) e a IDE Eclipse, versão 2.1.

5.4 Base de Dados do ASDA

O sistema gerenciador de banco de dados (SGBD) escolhido para o armazenamento das informações referentes aos usuário do ASDA foi o MySQL. O MySQL é um SGBD relacional de código fonte aberto, disponível para download em <http://www.mysql.com/>, e foi escolhido por oferecer as melhores vantagens quando comparado com as diversas implementações de sistemas de gerenciamento de banco de dados relacionais existentes no mercado, incluindo Oracle, Microsoft SQL Server, PostgreSQL e Informix (Tabela 5.1 (Suchring, 2002)).

Tabela 5.1: Comparação do MySQL com outros produtos de banco de dados relacional

Recurso	MySQL	Oracle	MS SQL Server	PostgreSQL
Transacional	Sim	Sim	Sim	Sim
De código-fonte aberto	Sim	Não	Não	Sim
Custo de propriedade total	Baixo	Alto	Alto	Baixo
Linguagens de desenvolvimento	Muitas	Muitas	Poucas	Muitas
Base de usuários corporativos	Sim	Sim	Sim	Não
Suporte da empresa	Sim	Sim	Sim	Não
Opera em diversas plataformas	Sim	Sim	Não	Sim

Todos os tipos de campos necessários para a criação das tabelas do ASDA, definidos em (Bruschi, 2002) são comportados pelo MySQL, assim como as funções necessárias para a manipulação de dados.

A execução de todos os comandos do MySQL foram executados via linha de comandos.

Tabela 5.2: Estrutura da tabela usuario

Campo	Tipo	Nulo	Chave Primária	Valor Padrão	Extra
idUsuario	int(11)		PRI	0	
nomeUsuario	varchar(75)	YES		NULL	
senha	varchar(6)	YES		NULL	
tipo	char(1)	YES		NULL	
tipoModelo	char(1)	YES		NULL	
Abordagem	char(1)	YES		NULL	
linguagem	varchar(15)	YES		NULL	
tamanhoLote	int(11)	YES		0	
tamWarmUp	int(11)	YES		0	
plataformaUsuario	varchar(15)	YES		NULL	

O primeiro passo para criar o banco de dados do ASDA foi definir a base de dados `asda`. O próximo passo foi a criação da tabela `usuario`. As linhas 1 e 2 da Figura 5.1 exibem a criação da base de dados e a conexão na base, respectivamente, e a linha 3 mostra a definição da tabela `usuario`.

```
CREATE TABLE usuario (
  idUsuario int(11) not null primary key,
  nomeUsuario varchar(75),
  senha varchar(6),
  tipo char(1),
  tipoModelo char(1),
  Abordagem char(1),
  linguagem varchar(15),
  tamanhoLote int(11),
  tamWarmUp int(11),
  plataformaUsuario varchar(15) );
```

Figura 5.1: Tabela de Usuários do ASDA

A tabela `usuario` agrupa os dois níveis de usuários do ASDA: usuário de simulação e super-usuário. A distinção entre os dois tipos de usuário é feito pelo campo `tipo`. A Tabela 5.2 exibe a estrutura da tabela `usuario`.

O campo `idUsuario` identifica unicamente cada usuário no ASDA. O campo `nomeUsuario` e `senha` são utilizados para a autenticação do usuário no ambiente. `tipo` especifica qual é o tipo de usuário: usuário de simulação ou super-usuário. O campo `tipoModelo` indica qual é o tipo de especificação que o usuário utiliza. O valor que o campo `abordagem` armazena qual o tipo de abordagem de simulação o usuário utiliza: seqüencial, MRIP, SRIP-CMB, SRIP-TimeWarp ou SRIP-CMB/TW. O campo `linguagem` indica qual a biblioteca/extensão funcional de simulação utilizada. Os campos `tamanhoLote` e `tamWarmUp`

representam, respectivamente, os valores para o tamanho do lote para a aplicação do método "Batch Means" e o tempo em que a simulação zera todos os contadores e medidas obtidas no programa de simulação e passa a ser o início do intervalo de medida. O campo `plataformaUsuario` indica qual é a plataforma de hardware que a simulação irá executar.

5.5 Definição das Estrutura de Dados do Modelo

A especificação gráfica e os parâmetros do modelo fornecidos pelo usuário compreendem a estrutura de dados que representa o modelo especificado. Essas informações são armazenadas nesse arquivo. Para recuperar o modelo e os seus parâmetros é necessário apenas ler as informações em arquivo. Dessa forma, o módulo de Interface Gráfica com o Usuário só necessita gerenciar as informações contidas no arquivo que recebe o nome do modelo e tem a extensão `.mod`.

Como Java é uma linguagem orientada a objetos, cada elemento do modelo é um objeto e cada objeto foi definido em uma classe. Cada classe contém os atributos correspondentes aos objetos e os métodos que a classe implementa.

Para o modelo foram definidos três classes: `Graph`, `Node` e `Arc`. A Figura 5.2 apresenta o diagrama de classes para a representação do modelo.

Figura 5.2: Diagrama de Classes do Modelo

O arquivo com a especificação do modelo contém todos os atributos presentes nas classe **Graph**, **Node** e **Arc**. Essas classes tornam possível recuperar o desenho do modelo exatamente como foi especificado no editor gráfico, os parâmetros do modelo e dos recursos. A Figura 5.3 apresenta a representação gráfica de um modelo no ASDA e a Figura 5.4 mostra como é a organização interna do arquivo gerado.

Figura 5.3: Exemplo de Representação Gráfica

Figura 5.4: Estrutura de Dados do Modelo

A classe **Graph** armazena os templates e todas as informações do modelo. A classe **Node** representa cada template do modelo (**Graph**). E a classe **Arc** é responsável por armazenar as ligações entre os templates do modelo.

5.5.1 *Graph*

A classe **Graph** contém as informações do modelo e todos os nós que compõem o modelo, que são os atributos da classe. A seguir são exibidos todos os atributos da classe **Graph** e os seus respectivos tipos:

idUsuario - Identificação do usuário "dono" do modelo.

nomeModelo - Nome do modelo.

tempoExecucao - Tempo de execução da simulação do modelo.

numeroCiclos - Número de ciclos.

tamanhoBatch - Tamanho do batch.

numeroMaximoEntidades - Número máximo de entidades.

tipoModelo - Tipo de modelo.

warmUp - Armazena se o warmup é definido pelo usuário ou automático.

nodes - Vetor que armazena os templates (fonte, centro de serviço e sorvedouro).

Optou-se por utilizar a classe **Vector** para armazenar os templates porque ela expande o seu tamanho conforme a necessidade. Desse modo não existe a possibilidade de alocar espaço em memória maior ou menor que o necessário. Sempre será alocado a quantidade exata para armazenar os templates do modelo. Assim, **nodes** foi definido criando um vetor *default*, não contendo elementos e a sua capacidade é incrementada em 1 sempre que um novo elemento é adicionado.

5.5.2 *Node*

A classe **Node** contém todos os atributos para a representação dos templates que compõem o modelo e os métodos que manuseiam a classe. Os templates representam a fonte geradora de clientes, centros de serviço e sorvedouro.

Foi implementada uma única classe para representar todos os templates. O que diferencia cada template é o atributo **tipoNo**. Dependendo do tipo de template os atributos serão ou não preenchidos. As listas a seguir ilustram os atributos da classe e quais os atributos específicos de cada template.

Presente em todos os templates

img - Indica qual a imagem que representa o template que foi inserido no modelo.

x - Representa em qual coordenada x a imagem foi inserida.

y - Representa em qual coordenada y a imagem foi inserida.

tipoNo - Indica qual o tipo de template que foi inserido no modelo.

idNo - Identifica o template com uma chave única.

nomeCentroServico - Representa cada template do modelo através de um nome.

arcs - Vetor que armazena as arestas de cada template.

Presente apenas no template Fonte

distribuicaoChegada - Indica qual a distribuição de chegada dos clientes.

mediaFonte - Tempo médio entre chegadas de novos clientes ao sistema.

Presente apenas no template centro de serviço

numServidores - Define o número de servidores do centro de serviço.

distribuicaoServico - Indica qual a distribuição de serviço para cada centro de serviço.

tempoResposta - Indica qual o parâmetro analisado durante a simulação para o centro de serviço. Cada centro de serviço pode ter um ou mais parâmetros.

throughput - Indica qual o parâmetro analisado durante a simulação para o centro de serviço. Cada centro de serviço pode ter um ou mais parâmetros.

tamanhoFila - Indica qual o parâmetro analisado durante a simulação para o centro de serviço. Cada centro de serviço pode ter um ou mais parâmetros.

estatisticaFilaVazia - Indica se serão geradas estatísticas da porcentagem de vezes em que o cliente encontrou fila vazia.

comprimentoMaxMin - Indica se serão geradas estatísticas para o cálculo do número de clientes em cada fila.

numFilas - Define o número de filas do centro de serviço. Se o recurso tiver várias filas em paralelo.

sequenciaFonte - Indica qual a sequência a ser utilizada na geração do número aleatório.

numVoltas - Indica o número de vezes que o usuário deve passar por um ciclo, se existir.

primRec - Indica se o centro de serviço é o primeiro recurso do modelo.

prob - Indica se a escolha do próximo recurso é por probabilidade.

ciclo - Indica se a escolha do próximo recurso é por número de voltas.

As coordenadas **x** e **y** indicam em qual posição na janela de edição o template está posicionado, como os templates podem ser arrastados dentro do editor, qualquer movimentação deles alteram os valores de **x** e **y**. As ligações entre os templates são armazenadas em um vetor de arestas (classe **Arc**). Cada template que é a origem de uma aresta possui um vetor de arestas que indicam o(s) destino(s) das ligações. A próxima seção apresenta em detalhes a classe **Arc**.

5.5.3 *Arc*

A classe **Arc** armazena o identificador do template de origem e do template de destino. Assim é possível determinar quais templates possuem ligações entre si. A lista abaixo exhibe os atributos definidos na classe **Arc**.

begin - Indica qual é o template de origem da aresta.

end - Indica qual é o template de destino da aresta.

idNo - Indica qual a identificação do template de destino da aresta.

tipoNo - Indica qual o tipo do template de destino da aresta.

probabilidade - Indica qual a probabilidade do template de origem ir para o template de destino.

5.6 *A Interface Gráfica do ASDA*

A interface gráfica do ASDA permite que o superusuário interaja com o ambiente e que os usuários de simulação especifiquem os seus modelos de simulação ou utilizem um programa de simulação pronto.

A interface desenvolvida permite ao superusuário:

- cadastrar novos usuários
- editar dados dos usuários
- excluir dados dos usuários
- pesquisar as informações dos usuários
- controlar o acesso ao ambiente através do gerenciamento de senhas dos usuários
- cadastrar as informações sobre o ambiente (tipos de técnicas de modelagem, abordagens de simulação distribuída, linguagens/extensões funcionais disponíveis no ASDA, tamanhos padrões de warm-up, tamanhos dos lotes e configurações de hardware)

Para o usuário de simulação a interface desenvolvida permite que ele:

- realize a especificação gráfica do modelo
- parametrize o modelo
- gere o programa de simulação
- visualize o programa de simulação

Quando o ASDA é executado ele se conecta com a base de dados `asda` e a primeira janela apresentada é a janela de autenticação do usuário. Nesta tela o usuário fornece o seu nome de usuário e a sua senha. O ASDA verifica se o usuário está cadastrado. Se o usuário está cadastrado uma janela de saudação aparecerá para ele de acordo com o seu tipo de usuário: superusuário ou usuário de simulação. As Figuras 5.5, 5.6, 5.7 mostram as janelas de autenticação do usuário, saudação para o superusuário e para o usuário de simulação, respectivamente.

Figura 5.5: Janela de Autenticação do Usuário no ASDA

Figura 5.6: Janela de Saudação para o SuperUsuário

Figura 5.7: Janela de Saudação para o Usuário de Simulação

Se o usuário é um superusuário, após a janela de saudação da Figura 5.6, é exibido uma janela (Figura 5.8) em que o superusuário configura as informações do usuário. Se o usuário for um usuário de simulação, após a janela mostrada na Figura 5.7, é exibido o editor gráfico do ASDA.

Figura 5.8: Cadastro de Usuário

5.6.1 Editor Gráfico

O editor gráfico do ASDA constitui o sub-módulo de especificação gráfica. O objetivo do editor gráfico é proporcionar ao usuário de simulação um ambiente para a definição do

modelo a ser simulado, permitindo que o usuário utilize todos os recursos disponíveis para representar seu sistema através de um modelo e possibilitando a geração completa de um programa de simulação.

O editor gráfico oferece um ambiente para que o modelo e os seus parâmetros sejam capturados no ASDA utilizando-se de uma interface amigável. A interface do editor gráfico é constituída por duas barras de ferramentas e uma barra de menu que disponibilizam ao usuário de simulação todas as funcionalidades do editor. Essas funcionalidades auxiliam o usuário a especificar o modelo, a parametrizar e finalmente a gerar o modelo.

O editor gráfico foi construído para modelar sistemas utilizando a técnica de representação gráfica *queuing statecharts*, mas a inclusão de novas técnicas é facilmente suportada. Isto é possível porque todos os métodos de acesso ao modelo e seus parâmetros podem ser utilizados através das classes do ASDA. O editor possibilita que o usuário de simulação especifique o modelo, remova elementos do modelo, salve e recupere modelos já definidos.

As informações coletadas são armazenadas em um arquivo. O módulo Gerador emprega as estruturas de dados na geração do programa de simulação. O capítulo 6 apresenta as informações sobre o módulo Gerador.

A verificação da consistência da especificação gráfica e dos parâmetros é realizada pelo módulo Avaliador Nível 1. Essa verificação é feita automaticamente pelo ASDA. Conforme o usuário de simulação vai modelando ou parametrizando o sistema a avaliação é realizada. A seção 5.6.3 descreve o funcionamento do módulo Avaliador Nível 1. Assim, o ASDA não permite que modelos inconsistentes gerem programas de simulação. A Figura 5.9 exibe a janela de edição do editor gráfico do ASDA.

A Janela de Especificação Gráfica

A janela de especificação gráfica é a área onde o usuário desenha o modelo do sistema. Ela ocupa a maior parte da janela do editor gráfico (área branca da Figura 5.9). Com o uso do mouse o usuário obtém acesso as barras de ferramentas e de menu para conceber, parametrizar e manipular o seu modelo. A parametrização dos recursos do modelo também pode ser feita dando um duplo clique em cima do recurso desejado.

As Barras de Ferramentas

O editor gráfico possui duas barras de ferramentas. Uma barra de ferramentas localizada na posição horizontal e na parte superior da janela de especificação gráfica (Figura 5.10) e outra vertical localizada a esquerda da janela de especificação gráfica (Figura 5.11). As barras de ferramentas são compostas por ícones que auxiliam o usuário de simulação a

Figura 5.9: Janela de Edição Gráfica

especificar o modelo de simulação.

Figura 5.10: Barra de Ferramentas Horizontal

A barra de ferramentas horizontal (Figura 5.10) é uma barra móvel que possibilita ao usuário de simulação arrastá-la e colocá-la na posição que desejar. Ela é composta de 4 ícones.

O ícone 1a possibilita ao usuário criar um novo modelo. Se o usuário de simulação está especificando um modelo quando ele comprime o ícone 1a é exibido uma janela de diálogo que pergunta se ele deseja salvar o modelo. Caso o usuário já tenha salvado as alterações do modelo não é apresentado nenhuma janela.

O próximo ícone é o 2a que permite que o usuário recupere um modelo já especificado. Um usuário de simulação pode recuperar modelos de outros usuários, porém sem a permissão de salvar alterações neste modelo, a menos que salve o modelo com outro nome.

O ícone 3a permite que o usuário salve a especificação gráfica e toda a parametrização do modelo. O usuário de simulação informa o nome do modelo e o diretório onde o arquivo será salvo.

O ícone seguinte é o 4a que é responsável pela parametrização do modelo. A parametrização do modelo será detalhada na seção 5.6.2.

A barra de ferramentas vertical (Figura 5.11) é uma barra fixa que integra as primitivas de desenho e seleção que permitem ao usuário de simulação a especificação gráfica do modelo.

Figura 5.11: Barra de Ferramentas Vertical

O ícone 1b representa o template fonte. O ícone 2b representa o template para centro de serviço. O ícone 3b representa o template sorvedouro e o ícone 4b possibilita a ligação entre dois recursos e as ramificações das mesmas. O ícone 5b é o responsável pela exclusão dos templates do modelo ou ligações.

A Barra de Menus

A barra de menus oferece os seguintes menus:

- Arquivo
- Parametrizar
- Gerador
- Visualizar
- Ajuda

O menu **Arquivo** disponibiliza os seguintes submenus:

- Novo - abre uma nova janela de especificação
- Abrir - abre um modelo já especificado
- Salvar - salva a especificação gráfica do modelo e a sua parametrização
- Sair - sai do ASDA

O menu **Parametrizar** apresenta os submenus:

- Modelo - parametriza o modelo
- Recurso - parametriza o recurso selecionado

O menu **Gerador** disponibiliza o submenu *Gerar* que gera o código do programa de simulação. O menu *Visualizar* apresenta o submenu *Código* que permite somente a visualização do código fonte do programa de simulação gerado. Para que o código fonte gerado seja alterado o modelo e seus parâmetros devem ser modificados ou é possível a edição do código em qualquer editor de texto, porém essas alterações não serão refletidas no modelo. O menu *Ajuda* apresenta os submenus:

- Ajuda - disponibilizará informações de ajuda sobre a utilização do ASDA
- Sobre - apresenta informações sobre versões do ASDA e o grupo de pesquisa

Dos itens citados acima, apenas o submenu *Ajuda* não está implementado.

5.6.2 Parametrização do Modelo de Simulação

A partir de uma análise da extensão funcional SMPL (Macdougall, 1987) foram identificados os parâmetros necessários para especificar um modelo de simulação completo. Esses parâmetros dividem-se em parâmetros do modelo e parâmetros específicos de cada recurso.

Os parâmetros do modelo são :

- Nome do modelo - nome atribuído ao modelo que foi especificado. Compreende os caracteres alfanuméricos e o caracter "_".
- Tempo de execução - tempo de simulação do modelo. Corresponde a um valor real. O valor 0.0 determina que a simulação não é limitada pelo tempo.

- Número máximo de entidades - fixa a simulação pelo número de clientes que deixam o sistema, para os modelos fechados. Corresponde a um valor inteiro positivo. O valor 0.0 determina que o sistema é um sistema aberto. Esse campo é inicializado com 0.0 automaticamente.
- Tempo de warmup - tempo em que a simulação zera todos os contadores e medidas obtidas no programa de simulação e passa a considerar os resultados obtidos. Pode ser utilizado para descartar dados obtidos enquanto o modelo não atingiu estado estacionário (Spolon, 1994).
- Tamanho de batch - determina o tamanho do lote para a aplicação do método "Batch Means"(não implementado). corresponde a um número inteiro positivo. O valor 0 indica que o método não será aplicado.
- Número de clientes - indica o número de clientes que chegam no sistema. Compreende um valor inteiro positivo.
- Tempo de chegada - indica o tempo da chegada do primeiro cliente ao sistema simulado. Corresponde a um valor real.

Os parâmetros específicos de cada recurso são os seguintes:

- Nome - nome do recurso. Compreende os caracteres alfanuméricos e o caracter "_".
- Número de servidores - número de servidores do centro de serviço do recurso. Compreende um valor inteiro positivo.
- Número de filas - número de filas do centro de serviço do recurso. Compreende um valor inteiro positivo.
- Estatística das filas
 - Fila vazia - determina se o usuário deseja conhecer ou não a porcentagem de vezes que o cliente encontrou a fila vazia. Corresponde a um valor *true* ou *false*, isto é, *true* se o usuário marcou o *check box* e *false* se o usuário não marcou.
 - Comprimento Max/Min - indica se o usuário deseja ou não saber estatísticas sobre o tamanho máximo e mínimo da fila. Usado somente quando o centro de serviço possuir o número de filas igual a um. Compreende um valor *true* ou *false*, isto é, *true* se o usuário marcou o *check box* e *false* se o usuário não marcou.

- Escolha do próximo recurso - indica qual o tipo de escolha para o próximo recurso se a saída de um recurso está ligada a vários outros recursos. A escolha pode ser de dois tipos:
 - Probabilidade - se o recurso está ligado a apenas um recurso ou a vários outros a escolha é por probabilidade. Compreende um valor real positivo.
 - Ciclo - determina o número vezes que um cliente deve passar por um recurso. Corresponde a um valor inteiro positivo. recurso.
- Parâmetro analisado - indica quais os parâmetros de desempenho serão avaliados durante a execução da simulação. Não está implementado. Os parâmetros podem ser:
 - Tempo de resposta
 - Throughput
 - Tamanho da fila

O ASDA obtém os parâmetros do modelo por meio dos menus e janelas de diálogo. As Figuras ?? e ?? apresentam as janelas de diálogo para a captura dos parâmetros do modelo.

5.6.3 Consistência do Modelo - Módulo Avaliador Nível 1

O nível 1 do módulo avaliador do ASDA é o responsável pela verificação da consistência do modelo especificado pelo usuário e atua em conjunto com o módulo de especificação gráfica. Ele também é o responsável por garantir que todas as informações fornecidas serão armazenadas. Isto é, ele não permite que quaisquer alterações no modelo sejam salvas sem que o usuário consinta.

A medida que o modelo é especificado a consistência do modelo é verificada. Cada novo recurso, ligação ou parâmetro introduzido no modelo é analisado. Assim, o ASDA não permite que modelos inconsistentes gerem programas de simulação.

Como o módulo avaliador nível 1 avalia a especificação gráfica e a parametrização do modelo, esse módulo foi dividido em dois sub-níveis:

- 1A** responsável pela verificação da consistência da especificação gráfica do modelo.
- 1B** responsável por verificar se todos os parâmetros foram fornecidos, se a os tipos, valores e intervalos estão coerentes e também por alertar o usuário sobre situações em que os valores podem levar a resultados incoerentes na simulação.

De acordo com a especificação *queuing statecharts* algumas regras devem ser seguidas no processo de avaliação do nível 1A:

- Somente uma fonte geradora de clientes (template fonte) é permitido para cada modelo
- Nenhuma ligação é permitida entre uma fonte e um sorvedouro, o contrário também é válido
- Todos os templates presentes no modelo devem estar conectados entre si. Templates sem conexão não são permitidos
- Um sorvedouro somente pode ser destino de uma ligação (aresta) e nunca origem
- Uma fonte sempre é origem de uma ligação e nunca destino

A Figura 5.12 mostra a atuação do módulo avaliador nível 1A alertando o usuário de que a ligação entre um centro de serviço e fonte não é permitida e a mensagem da Figura 5.13 não permite que o usuário parametrize o modelo, pois existem templates sem conexão.

A atuação do módulo avaliador nível 1A ocorre durante a especificação do modelo, isto é, toda vez que um elemento (template ou ligação) é adicionado ao modelo. O módulo avaliador nível 1B avalia se todos os campos com entradas obrigatórias foram preenchidos. Essa avaliação ocorre sempre que alguma informação é armazenada no arquivo de especificação do modelo ou quando o usuário aciona a opção para gerar o programa de simulação. Se algum parâmetro obrigatório não for preenchido o avaliador emite um alerta para o usuário, identifica o erro e não prossegue na geração do programa. A avaliação também implica na averiguação dos tipos de dados que cada parâmetro comporta. Por exemplo, se o tempo de *warmup* escolhido é o determinado pelo usuário e mais tarde sofrer a alteração para automático, o valor recebido é desconsiderado pelo avaliador nível 1B.

Figura 5.12: Módulo Avaliador Nível 1 - Verificação da Consistência da Especificação Gráfica

Quando a parametrização do modelo não está completa o ASDA emite um alerta, como exibido na Figura 5.14, indicando que existe(m) parâmetro(s) sem valor(es) especificado. Para cada um dos parâmetros a serem fornecidos pelo usuário o procedimento de verificação da consistência é executado. Dessa forma o usuário não deixará de fornecer

Figura 5.13: Módulo Avaliador Nível 1A - Verificação da Consistência da Especificação Gráfica

todos os parâmetros necessário para a geração do programa de simulação e um modelo inconsistente não será definido. Se o modelo está inconsistente o programa de simulação não é gerado.

Figura 5.14: Módulo Avaliador Nível 1B - Verificação da Consistência da Parametrização do Modelo

Quando o usuário deseja sair do ASDA e alguma informação não foi salva o ASDA lança um alerta (Figura 5.15) perguntando se o usuário deseja salvar o modelo.

Figura 5.15: Armazenamento do Modelo

5.7 Considerações Finais

Neste capítulo foram apresentados os aspectos relativos a implementação do módulo de interface com o usuário.

A escolha da técnica de especificação gráfica *Queuing Statecharts* mostrou-se uma especificação mais completa para sistema de filas. Porém outras técnicas para modelagem podem ser incluídas no ASDA e se necessário as classes já definidas podem ser alteradas

ou estendidas, através de herança simples suportada por Java, e novas classes podem ser incluídas para que todas as características da nova técnica sejam representadas e preservadas.

A ferramenta para a implementação do módulo de interface com o usuário e do módulo avaliador nível 1 utilizando a tecnologia Java e a ferramenta Eclipse permitem que o ASDA seja executado em uma variedade de sistemas operacionais e opere em conjunto com múltiplas linguagens de programação.

O desenvolvimento da interface gráfica do ASDA alcançou os objetivos propostos em (Bruschi, 2002) ao disponibilizar ao usuário fácil acesso às opções oferecidas pelo ambiente, bem com um editor gráfico para a definição do modelo a ser simulado.

O usuário escolhe quais componentes devem ser acrescentados ao modelo, quais devem ser removidos e por meio de um duplo clique escolhe quais componentes serão parametrizados. As janelas de diálogo requisitam os parâmetros dos recursos e do modelo. Sempre que alguma informação errada é fornecida ao ASDA o módulo Avaliador Nível 1 emite mensagens de alerta para o usuário informando o motivo.

A módulo de interface gráfica com o usuário produz um arquivo que contém os templates e ligações do modelo e os seus parâmetros.

Implementação do Gerador do Programa de Simulação

6.1 Considerações Iniciais

O gerador de programa de simulação do ASDA é a ferramenta que, a partir da especificação do modelo em *queuing statecharts*, transforma automaticamente essa especificação na implementação do programa de simulação. A especificação do modelo é fornecida pelo módulo de interface gráfica com o usuário que armazena as especificações do modelo em um arquivo. Essa especificação é utilizada pelo módulo Gerador que cria o programa de simulação na linguagem de programação alvo.

A utilização do gerador possibilita que o usuário forneça o modelo e os parâmetros do sistema a ser avaliado e obtenha o programa de simulação automaticamente, afastando assim do usuário a tarefa de transcrição do modelo em um programa de simulação.

O gerador desenvolvido implementa programas de simulação em SMPL (Macdougall, 1987), uma extensão funcional da linguagem C para simulação orientada a eventos.

O gerador do programa de simulação foi implementado na linguagem de programação Java, utilizando-se o compilador Java 2 *Standard Edition* Versão 1.4 da Sun *Microsystems* e a ferramenta Eclipse (Eclipse, 2001).

6.2 Gerador

O gerador é parte integrante do ASDA, cujo objetivo é oferecer ao usuário um ambiente automático para o desenvolvimento de simulação (Bruschi, 2002).

Para que o gerador gere os programas de simulação devem existir dois arquivos: um arquivo de gabarito e um arquivo contendo as informações do modelo e seus parâmetros. O programa gerador lê as instruções do arquivo de gabarito e executa os métodos correspondentes. Em um novo arquivo o programa de simulação é transcrito. O arquivo gabarito (Figura 6.1) contém trechos de códigos formatados na linguagem ou biblioteca alvo e comandos que indicam quais os métodos do gabarito devem ser executados.

```

/* ----- */
#include <stdio.h>
#include <conio.h>
#include "simpl.h"
#include "rand.h"

main()
{
    /* Fase de Definições */
    %0define tempo_max_max
    int Event = 1, Customer = 1, Aleatorio;
    %4gera_contadores_loop
    %1define variaveis

    /* Fase de Preparação */
    %2define simpl

    /* cria e da nome as facilidades */
    %3define server

    /* escalona a chegada do primeiro cliente */
    %4escalona_primeiro_evento

    %8gera_trace

    /* Fase de Implementação */
    %5gera_loop
    {
        %0warm_up
        %6gera_cause
        %7gera_switch
        {
            %8gera_eventos
        }
    }

    /* gera o relatorio da simulacao */
    %9gera_relatorio_padrao
    %0gera_relatorio_estatisticas_max_min
    %0gera_relatorio_estatisticas_fila_vazia
}
/* ----- */

```

Figura 6.1: Gabarito

O gerador distingue qual é o método que deve ser executado identificando qual linha possui o caracter "%". Reconhecida a linha ele recupera o próximo caracter após o "%" e identifica no gerador o método correspondente para ser executado. Os trechos de código do gabarito que não possuem o caracter "%" são copiados para o programa de simulação exatamente como se apresentam no gabarito. A leitura do arquivo de gabarito é sequencial. A Figura 6.2 exhibe o processo de leitura do arquivo de gabarito e a chamada ao método do gerador.

```

/* ----- */
#include <stdio.h>
#include <conio.h>
#include "smpl.h"
#include "rand.h"

main()
{
    /* Fase de Definições */
    %0define_tempo_max
    int Event = 1, Customer = 1, Aleatorio;
    %Agera_contadores_loop
    %1define_variaveis

    /* Fase de Preparação */
    %2define_smpl

    /* cria e da nome as facilidades */
    %3define_server

    /* escalona a chegada do primeiro cliente */
    %4escalona_primeiro_evento

    %Bgera_trace

    /* Fase de Implementação */
    %5gera_loop
    {
        %Gwarm_up
        %6gera_cause
        %7gera_switch
        {
            %8gera_eventos
        }
    }

    /* gera o relatorio da simulacao */
    %9gera_relatorio_padrao
    %Cgera_relatorio_estatisticas_max_min
    %Dgera_relatorio_estatisticas_fila_vazia
    }
/* ----- */

```

Figura 6.2: Trecho de Código da Leitura do Gabarito pelo Gerador

A Figura 6.3 exibe a estrutura de arquivos no módulo gerador e o fluxo de dados das informações.

O gerador dá origem ao programa de simulação utilizando as informações providas pelo editor gráfico. Essas informações estão disponíveis em um arquivo que contém a especificação gráfica do modelo e a sua parametrização.

O gerador é formado por um conjunto de métodos que são invocados durante o processo de geração do código do programa de simulação. Cada método é responsável pela construção de um trecho do código do programa de simulação. O gerador possibilita:

1. Gerar os principais comandos da biblioteca SMPL para:
 - preparação do modelo
 - definição e controle de recursos

Figura 6.3: Estrutura de Arquivos no Módulo Gerador

- escalonamento e geração de eventos
- geração de variáveis aleatórias
- depuração do código
- coleta de dados e emissão de relatórios

2. Gerar relatório padrão do SMPL com informações sobre:

- tempo de execução da simulação
- utilização dos recursos
- período médio ocupado
- comprimento médio da fila associada a cada recurso
- contadores de operações realizadas

3. Gerar novos relatórios com informações estatísticas sobre o número máximo e mínimo de clientes na fila e a porcentagem de vezes em que o cliente encontra uma determinada fila vazia.

6.3 Execução do Gerador e Geração do Código

Após o usuário fornecer o modelo do sistema que ele deseja simular e todos os parâmetros necessários o gerador está pronto para ser executado.

O gerador possui uma interface simples para a sua execução. A interface do gerador de programa de simulação é gráfica. O acesso a funcionalidade consiste em acionar a opção **Gerar** do menu **Gerador**, conforme apresentada na Figura 6.4.

Figura 6.4: Execução do Gerador do Programa de Simulação

Se por algum motivo a execução do gerador falhar uma mensagem é emitida ao usuário dizendo que o código não pode ser gerado. Porém, se o código foi executado com sucesso a mensagem de que o código foi gerado com sucesso é exibida. O programa gerado recebe o nome do modelo e a extensão `.cpp`. O ASDA disponibiliza a visualização do programa de simulação gerado através do menu **Visualizar**. O menu **Visualizar** não possibilita a alteração do programa gerado. Quaisquer alterações devem ser realizadas modificando o modelo e os seus parâmetros.

A geração do código é uma atividade realizada utilizando-se o arquivo que contém a especificação gráfica do modelo e a sua parametrização, o arquivo de gabarito e o gerador.

Todo o processo de geração do código é regulado por um método `gabarito( FileReader gabarito )`, contido na classe `Gerador`, que recebe como parâmetro um objeto `FileReader` (Sun, 1996) que contém o nome do arquivo de gabarito. O método `gabarito( FileReader gabarito )` identifica quais as primitivas que devem ser executadas

(precedidas pelo caracter %).

As primitivas que compõem o gerador estão presentes na classe `Gerador`. Cada trecho do programa de simulação é produzido pelos métodos da classe `Gerador` correspondentes a cada uma das primitivas.

O processo de geração de código no gerador envolve três fases: definição, preparação do modelo e implementação. A fase de definição gera as variáveis para a execução do programa de simulação. A fase de preparação do modelo gera as primitivas para preparação do ambiente e a fase de implementação define os eventos que compõem o programa de simulação. O gerador acessa o arquivo que contém o modelo e segue a sequência de execução dos métodos descritas no `Gabarito.dat`. Após a execução de cada método o gerador busca no gabarito a próxima instrução que será executada.

A seção seguinte mostra os métodos utilizados na geração de código para alguns comandos SMPL, as primitivas do arquivo de gabarito que disparam a geração do código e o trecho de código produzido para o modelo da Figura 6.5.

Figura 6.5: Modelo para Exemplificação do Código Gerado

6.3.1 Fase de Definições

defineTempoMax()

Gera o comando para limitar a execução da simulação: por tempo, por número de clientes que deixam o sistema ou pelo número de ciclos do cliente no sistema. Esse método é executado quando o gerador lê do arquivo de gabarito a primitiva `$\%$0define$_$tempo$_$num$_$max`. A execução do método produz a seguinte linha no programa de simulação:

```
float Te = 10000;
```


defineVarServer()

Gera as variáveis para as definições dos recursos que compõem o modelo a ser implementado quando a primitiva `$\%$1define$_$variaveis` é localizada no gabarito. Esse método produz o trecho de código a seguir:

```
int Server1, Server2;
```

defineVarTempos()

Gera as variáveis para os tempos médios entre chegadas e serviço. Esse método também é executado quando `$\%$1define$_$variaveis` é encontrada no gabarito. O resultado da execução do método é:

```
real Ta1 = 10, Ts1 = 10, Ta2 = 0, Ts2 = 10;
```

defineBMeans()

Define as variáveis necessárias para utilizar o método de análise Batch Means. O método `defineBMeans()` é executado quando a primitiva `\%1define$_$variaveis` é localizada no gabarito. Como resultado gera o seguinte trecho de código:

defineArqSaida()

Declara e abre para escrita o arquivo de saída no código do programa de simulação. Este arquivo de saída conterá o relatório da simulação. A sua execução ocorre quando a primitiva `\%1define$_$variaveis` é lida do gabarito pelo gerador e o resultado da execução do método é o trecho de código:

```
FILE *p, *saida;
saida = fopen("C:\thais\usp\Fonte\teste.out", "w");
if ((p = sendto(saida)) == NULL)
    printf("Erro na saida\n");
```

6.3.2 Fase de Preparação do Modelo

nomeParametro()

Gera o comando SMPL para a inicialização do modelo de simulação. Esse método é executado quando a primitiva `$\%2define$_$smp1` é localizada no gabarito. O resultado da sua execução produz o seguinte trecho de código:

```
smp1(0," Teste");
```

geraDefServer()

O método `geraDefServer()` cria e nomeia cada um dos recursos do modelo. Isto é, gera o comando `facility` para a definição de cada recurso que compõem o sistema. A sua execução ocorre quando a primitiva `$\%$define$_$server` é localizada. A execução do método produz:

```
Server1 = facility("Escalonador",1);
```

```
Server2 = facility("Processador",1);
```

6.3.3 Fase de Implementação

geraPrimeiroEvento()

Escalona o primeiro evento a ocorrer no caso de sistema com uma única entrada, ou todos os eventos que devem ser escalonados antes do início da simulação para modelos de sistemas fechados. Esse método é executado quando o gerador reconhece a primitiva `%4escalona_primeiro_evento` no gabarito. O resultado da execução do método é:

```
schedule(1,0, Customer);
```

geraLoop()

Gera o comando de repetição, limitando a simulação por: tempo, número de clientes que passam pelo sistema, número de voltas no sistema e método de análise Batch Means. Esse método é disparado quando a primitiva `%5gera_loop` é reconhecida no gabarito pelo gerador. O produto da execução desse método no programa de simulação é:

```
while ( (time() < Te) )
```

geraCause()

Implementa o comando **cause** quando a primitiva %6gera_cause é encontrada pelo gerador no gabarito. Produz como resultado o seguinte trecho de código:

```
cause(&Event,&Customer);
```

geraSwitch()

Gera o comando **switch**. A execução desse método ocorre quando o gerador lê no arquivo de gabarito a primitiva %7gera_switch e o resultado da sua execução é:

```
switch(Event)
```

geraEventos()

Gera os eventos para o escalonamento dos eventos que constituem a simulação. Esse método é executado pelo gerador quando a primitiva %8gera_eventos é lida do arquivo de gabarito pelo gerador. A execução desse método dispara os métodos **geraRelease()** e **geraEntradasSaida()**. O método produz como resultado:

```
case 1:
    schedule(2,0.0,Customer);
    schedule(1,expntl(Ta1),Customer);
    break;
case 2:
    if (request(Server1,Customer,0) == 0)
        schedule(3,expntl(Ts1),Customer);
    break;
case 3:
    release(Server1, Customer);
    schedule(4, 0.0, Customer);
    break;
case 4:
    if (request(Server2,Customer,0) == 0)
        schedule(5,expntl(Ts2),Customer);
    break;
case 5:
    release(Server2, Customer);
```

```
break;
```

geraRelease(), *geraEntradasSaidas()*

Criam os comandos de liberação do recurso que pode ser a entrada para um outro recurso ou a saída do sistema.

geraRelatorioFinal()

Gera o relatório final (padrão do SMPL) no programa de simulação com informações estatísticas sobre a simulação. Esse método é executado quando a primitiva %9gera_relatorio_padrao é localizada no gabarito. O resultado da execução do método é:

```
report();
```

geraRelatEstMaxMin()

Gera relatório com as estatísticas das filas dos recursos especificados pelo usuário. A sua execução ocorre quando o gerador lê no arquivo de gabarito a primitiva %9gera_relatorio_estatisticas_max_min e se o usuário de simulação assinalou a opção de gerar esse relatório.

geraRelatFilaVazia()

Gera relatório com as estatísticas das filas dos recursos especificados pelo usuário indicando a porcentagem de vezes na qual o cliente encontra a fila vazia. Esse método verifica se o usuário de simulação deseja gerar esse relatório. A primitiva que indica a execução do método é %9gera_relatorio_estatisticas_fila_vazia.

6.4 Exemplos de Programas Produzidos pelo Gerador

A seguir são apresentados alguns exemplos de programas de simulação gerados pelo ASDA e os modelos em queuing statecharts.

6.4.1 Exemplo 1

Para o exemplo 1 (Figura 6.6) a execução da simulação é limitada pelo tempo de execução e total de clientes que deixam o sistema.

Figura 6.6: Exemplo 1

Para o exemplo um os seguintes parâmetros foram definidos:

- Fonte
 - Distribuição: exponencial
 - Média: 10.0
 - Sequência: 0.0
- Escalonador
 - Distribuição: exponencial
 - Média: 10.0
 - Sequência: 0.0
- Processador
 - Distribuição: exponencial
 - Média: 10.0
 - Sequência: 0.0
 - Estatística: Fila Vazia
- Modelo
 - Nome modelo: teste1.mod
 - Tempo de Execução: 10000
 - Warmup: 0.0
 - Primeira Chegada: 0.0
 - Relatório Final: Sim

94CAPÍTULO 6. IMPLEMENTAÇÃO DO GERADOR DO PROGRAMA DE SIMULAÇÃO

O código do programa de simulação gerado é apresentado a seguir:

```
/*
-----
*/ #include <dir.h> #include <stdio.h> #include <conio.h> #include
"simpl.h" #include "rand.h"

main() {
    /* definicoes */
    float Te = 10000;
    int Event = 1, Customer = 1, Aleatorio;
    real Ta1 = 10, Ts1 = 10, Ta2 = 0, Ts2 = 10;
    int Server1, Server2;
    unsigned int Tot2 = 0, Vaz2 = 0;
    unsigned int Total_Clientes = 0;
    FILE *p, *saida;
    saida = fopen("teste.out","w");
    if ((p = sendto(saida)) == NULL)
        printf("Erro na saida\n");

    /* prepara o sistema de simulacao e da nome ao modelo */
    simpl(0," Teste");

    /* cria e da nome as facilidades */
    Server1 = facility("Escalonador",1);
    Server2 = facility("Processador",1);

    /* escalona a chegada do primeiro cliente */
    schedule(1,0, Customer);

    while ( (time() < Te) )
    {
        cause(&Event,&Customer);
```

```

    switch(Event)
    {
        case 1:
            schedule(2,0.0,Customer);
            schedule(1,expntl(Ta1),Customer);
            break;
        case 2:
            if (request(Server1,Customer,0) == 0)
                schedule(3,expntl(Ts1),Customer);
            break;
        case 3:
            release(Server1, Customer);
            schedule(4, 0.0, Customer);
            break;
        case 4:
            Total_Clientes = inq(Server2);
            if (Total_Clientes == 0)
                Vaz2++;
            Tot2++;
            if (request(Server2,Customer,0) == 0)
                schedule(5,expntl(Ts2),Customer);
            break;
        case 5:
            release(Server2, Customer);
            break;
    }
}

/* gera o relatorio da simulacao */
report();
fprintf(saida," \n\nRelatório - Total de vezes - Fila Vazia \n");
fprintf(saida, "\n Total clientes no recurso 2 : %d ",Tot2);
fprintf(saida,"\n Total clientes que encontraram fila vazia no recurso 2 : %d ",Vaz2);
fclose(saida);
} /*
-----

```

*/

O programa de simulação produzido pelo gerador foi executado corretamente e gerou o relatório final conforme mostra a Figura 6.7.

Figura 6.7: Exemplo 1

6.4.2 Exemplo 2

O exemplo 2 (Figura 6.8) especificou seqüências para geração de números aleatórios e informações sobre o total de clientes na fila de rec1.

Figura 6.8: Relatório Final Exemplo 2

Para o exemplo um os seguintes parâmetros foram definidos:

- Fonte
 - Distribuição: exponencial
 - Média: 20.0
 - Seqüência: 0.0
- recurso1
 - Distribuição: exponencial
 - Média: 10.0
 - Seqüência: 0.0
- recurso2
 - Distribuição: exponencial
 - Média: 15.0
 - Seqüência: 0.0
- recurso3
 - Distribuição: exponencial
 - Média: 20.0
 - Seqüência: 0.0
- Modelo
 - Nome modelo: m2.mod
 - Tempo de Execução: 1000
 - Warmup: 0.0
 - Primeira Chegada: 0.0
 - Relatório Final: Sim

O código gerado segundo a parametrização acima é apresentado a seguir:

```
/*-----*/
#include <dir.h> #include <stdio.h> #include <conio.h>

#include "smpl.h" #include "rand.h"
```

```

main() {

    /* definicoes */
    float Te = 1000;
    int Event = 1, Customer = 1, Aleatorio;
    real Ta1 = 20, Ts1 = 10, Ta2 = 0, Ts2 = 15,
        Ta3 = 0, Ts3 = 20;
    int Server1, Server2, Server3;
    unsigned int Max1 = 0, Min1 = 1000;
    unsigned int Total_Clientes = 0;

    FILE *p, *saida;
    saida = fopen("m2.out","w");
    if ((p = sendto(saida)) == NULL)
        printf("Erro na saida\n");

    /* prepara o sistema de simulacao e da nome ao modelo */
    smpl(0," m1");

    /* cria e da nome as facilidades */
    Server1 = facility("rec1",1);
    Server2 = facility("rec2",1);
    Server3 = facility("rec3",1);

    /* escalona a chegada do primeiro cliente */
    schedule(1,0, Customer);

    while ( (time() < Te) )
    {
        cause(&Event,&Customer);
        switch(Event)
        {
            case 1:
                schedule(2,0.0,Customer);
                schedule(1,expntl(Ta1),Customer);
                break;

```

```

case 2:
    Total_Clientes = inq(Server1);
    if (Total_Clientes > Max1)
        Max1 = Total_Clientes;
    else
        if (Total_Clientes < Min1)
            Min1 = Total_Clientes;
    if (request(Server1, Customer, 0) == 0)
        schedule(3, expntl(Ts1), Customer);
    break;
case 3:
    release(Server1, Customer);
    schedule(4, 0.0, Customer);
    break;
case 4:
    stream(5);
    if (request(Server2, Customer, 0) == 0)
        schedule(5, expntl(Ts2), Customer);
    break;
case 5:
    release(Server2, Customer);
    schedule(6, 0.0, Customer);
    break;
case 6:
    if (request(Server3, Customer, 0) == 0)
        schedule(7, expntl(Ts3), Customer);
    break;
case 7:
    release(Server3, Customer);
    break;
}
}

/* gera o relatorio da simulacao */
report();
fprintf(saida, " \n\nRelatório - Máximo e Mínimo das Filas \n ");
fprintf(saida, "\n Maximo clientes Recurso 1 : %d ", Max1);

```

```

    fprintf(saida, "\n Minimo clientes Recurso 1 : %d ",Min1);
    fclose(saida);
}

/*-----*/

```

O programa de simulação produzido pelo gerador foi executado corretamente e gerou o relatório final conforme mostra a Figura 6.9.

Figura 6.9: Relatório Final Exemplo 2

6.5 Considerações Finais

Este capítulo apresentou aspectos sobre a implementação de um gerador de programas de simulação. O gerador desenvolvido é parte integrante do ASDA.

Os programas produzidos pelo gerador são escritos na extensão funcional SMPL, utilizando modelos descritos em *queuing statecharts*.

O gerador usa as informações contidas na especificação do modelo e no arquivo de gabarito para produzir os programas de simulação. O módulo de interface com o usuário, através do editor gráfico fornece a especificação do modelo e o gabarito disponibiliza toda a organização de um programa de simulação SMPL. O gabarito é o responsável por coordenar a execução do gerador.

O próximo capítulo apresenta as conclusões gerais obtidas com o desenvolvimento desta dissertação.

Conclusão

7.1 Conclusões Gerais

A revisão bibliográfica apresentada nos Capítulos 2 e 3 apresentaram os conceitos necessários sobre avaliação de desempenho e as técnicas para a representação de modelos. O Capítulo ?? apresentou o ambiente para simulação distribuída ASDA (Bruschi, 2002).

A escolha da técnica para a avaliação de desempenho compreende uma etapa fundamental no processo de avaliação de desempenho. Diferentes técnicas podem ser utilizadas para a avaliação de desempenho. A técnica de aferição é a que oferece resultados mais precisos. Porém, não pode ser aplicada quando o sistema que se deseja avaliar não existe ou quando se deseja uma avaliação de desempenho sem a influência da própria aferição. As técnicas de modelagem representam uma abstração do modelo que contém as características essenciais do sistema sem a necessidade de realizar a experimentação do mesmo.

As técnicas de modelagem requerem a solução do modelo. Essa solução pode ser obtida utilizando-se a solução analítica ou a simulação. A solução analítica produz resultados com esforço computacional menor. Porém, a casos em que o modelo pode ser tornar muito complexo e nesses casos as simplificações colocadas para representar o modelo tendem a comprometer a exatidão da solução e alterações no sistema podem significar a reconstrução de todo o modelo. A simulação é uma técnica muito útil e flexível. Permite que quaisquer alterações no sistema sejam refletidas facilmente no programa de simulação. Tanto pode ser usada para a avaliação de sistema que não existem quanto para sistemas onde a sua instrumentação pode perturbar a sua utilização.

Foram apresentados no Capítulo 3 as três técnicas de modelagem mais citadas na literatura e duas novas abordagens. Redes de fila são muito úteis quando se deseja representar

sistemas que possuam apenas os elementos fila e servidor. Mas quando se deseja representar os estados em que o sistema se encontra e as suas transições no transcorrer do tempo redes de fila se mostra uma representação limitada. As redes de Petri e *Statecharts* suprem essa limitação das redes de fila, contudo redes de Petri não representam adequadamente modelos paralelos. Os *Statecharts* não conseguem representar situações em que existem a formação de filas nem contemplam a individualização dos clientes no sistema.

As especificações *Statecharts* Estocásticos e *Queuing Statecharts* possuem um conjunto de templates que representam os componentes básicos de um sistema de filas. Como as duas técnicas se baseiam nos *Statecharts* elas conseguem representar características dos sistemas como: paralelismo, hierarquia, mecanismos de comunicação, etc. Mas *Statecharts* Estocásticos não conseguem representar o caminho linear que os cliente seguem no sistema, o que é suprido em *Queuing Statecharts*.

O Capítulo 4 apresentou o ASDA, que é um ambiente de desenvolvimento de simulações. O ASDA permite que usuários com ou sem muita experiência especifiquem seus modelos e gerem programas de simulação. Enquanto o modelo é especificado o usuário dispõe de um mecanismo que realiza a verificação e validação do modelo e os seus parâmetros. Quando o modelo apresentar inconsistências o ASDA emite mensagens de alerta especificando o tipo do erro. A utilização do ASDA prove agilidade no processo de desenvolvimento de programas de simulação, conseqüentemente facilita a tarefa de modelagem e avaliação de desempenho dos sistemas.

7.2 Contribuições do Trabalho

A seguir são apresentadas as contribuições obtidas com o desenvolvimento desse trabalho:

- Uma análise crítica da bibliografia existente no que se refere a avaliação de desempenho e as técnicas de modelagem. Identificando as características, vantagens e desvantagens de cada técnica de modelagem e solução do modelo
- Desenvolvimento do Módulo de Interface com o Usuário do ASDA que abrangeu a construção de um editor gráfico que permite a especificação gráfica do modelo e a disponibilização de modelos já especificados, um gerenciador de usuários do ambiente que configura as preferências e atribui os privilégios que compete a cada usuário e armazena em uma base de dados
- Implementação do Módulo de Interface com o Usuário independente da linguagem ou pacote de simulação

- Especificação das informações e parâmetros necessários para a geração de um programa de simulação
- Implementação do Módulo Avaliador Nível 1 e definição das regras para a verificação da consistência da especificação gráfica do modelo e a validação dos parâmetros do modelo
- Implementação do Gerador de Programas de Simulação que possibilita a geração automática do código do programa de simulação na extensão funcional SMPL
- Possibilidade de utilização de modelos já definidos por outros usuários e realização de adaptações nesses modelos
- Inclusão no ASDA de uma nova abordagem para a especificação de modelos de sistemas: *Queuing Statecharts*
- Facilidade de inserção de novas técnicas de especificação de modelos devido ao fato dos Módulos de Interface com o Usuário e Avaliador e o Gerador de Programas de Simulação terem sido implementados utilizando o conceito de orientação a objetos, presentes na linguagem Java, assim novas classes podem ser estendidas ou construídas a partir das existentes. Além disso toda a documentação incluída nas classes formam a API (*Application Programming Interface*) do ASDA. Com isso todos os métodos e atributos de classe, bem como as próprias classes possuem documentação gerada automaticamente que facilita o trabalho de adição de novas funcionalidades, correção de defeitos e integração com outros recursos
- Esta pesquisa gerou a seguinte publicação:
 - BRUSCHI, S. M., SANTANA, R. H. C., SANTANA, M. J., AIZA, T. S. AN AUTOMATIC DISTRIBUTED SIMULATION ENVIRONMENT In: 2004 Winter Simulation Conference, (WSC'2004) Washington, D.C., United States of the America.

7.3 Sugestões para Trabalhos Futuros

A seguir são listadas algumas sugestões para trabalhos futuros:

- Implementação do gerenciador de informações sobre o ambiente (informações de hardware, tipos de técnicas de modelagem, abordagens de simulação, linguagens/extensões funcionais, tamanhos padrões de warmup e tamanhos dos lotes) e também as informações sobre os Módulos Avaliador (níveis 2 e 3), Replicador, Executor e Escalonamento

- Implementação de um gerenciador de um repositório de programas de simulação
- Inclusão da funcionalidade para gerar o programa de simulação executável
- Inclusão de mais linguagens/extensões funcionais para o Módulo Gerador
- Construção do guia de ajuda para o usuário
- O editor gráfico pode permitir a concatenação de dois modelos já especificados
- A inclusão de novas técnicas de especificação gráfica, como por exemplo rede de filas
- Extensão do *Queuing Statecharts* para representar a posse simultânea de recursos
- Pode-se acrescentar ao editor gráfico a possibilidade de visualizar o código correspondente a um determinado ponto do modelo
- Inclusão de novas distribuições e a implementação de novas estatísticas
- Integração dos módulos implementados com os demais módulos propostos na especificação do ASDA (Bruschi, 2002)

Referências Bibliográficas

- Ajmoné Marsan, M.; Balbo, G.; Conte, G. (1986). *Performance Models of Multiprocessor Systems*. The Mit Press, Cambridge.
- Ajmoné Marsan, M.; Bobbio, A.; Donatelli, S. (1998). *An Introduction Lecture Notes in Computer Science*.
- Allen, A. O. (1990). *Probability, Statistics, and Queueing Theory*. Academic Press.
- Aries, J. A.; Banerjee, S.; Brittan, M. S.; Dillon, E.; Kowalik, J. S.; Lixvar, J. P. (2002). Capacity and performance analysis of distributed enterprise systems. *Commun. ACM*, v.45, n.6, p.100–105.
- Ayani, R. (1998). *Parallel Simulation*. Lecture Notes in Computer Science, n. 729.
- Balsamo, S.; Inverardi, P.; Mangano, C. (1998). An approach to performance evaluation of software architectures. *Proceedings of the First International Workshop on Software and Performance*, p. 178 – 190, Santa Fe, New Mexico, United States of the America.
- Bruschi, S. M. (2002). *ASDA - Um Ambiente de Simulação Distribuída Automático*. Tese (Doutorado), ICMC-USP.
- Bryant, R. E. (1977). Simulation of packet communications architecture computer. Relatório Técnico 188, Massachusetts Institute of Technology, LCS, Massachusetts Institute of Technology.
- Chandy, K. M.; Misra, J. (1979). Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering*, v.SE-5, n.5, p.440–452.
- Couto, C. L. S. (1999). Um estudo comparativo de benchmarks paralelos. Dissertação (Mestrado), Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo.
- Eclipse (2004). Eclipse foundation. <http://www.eclipse.org>.

- Francês, C. R. L. (2001). *Statecharts Estocásticos e Queuing Statecharts: Novas Abordagens para Avaliação de Desempenho Baseadas em Especificação Statecharts*. Tese (Doutorado), ICMC-USP.
- Francês, C. R. L.; Santana, M. J.; R. H. C. Santana, N. L. V.; Solon, V. C. (2000a). Stochastic extension to statecharts for representing performance models: an application to a file system. *Simpósio Brasileiro de Arquitetura de Computadores e Computação de Alto Desempenho (SBAC-PAD)*, p. 365 - 372, São Pedro, São Paulo, Brasil.
- Francês, C. R. L.; Santana, M. J.; R. H. C. Santana, N. L. V.; Solon, V. C. (2001). Statecharts estocásticos e queuing statecharts: Novas abordagens para avaliação de desempenho baseadas em especificação statecharts. *15o Simpósio Brasileiro de Engenharia de Software (SBES2001)*, p. 316 - 331, Rio de Janeiro, Rio de Janeiro, Brasil.
- Francês, C. R. L.; Santana, M. J.; R. H. C. Santana, N. L. V.; Solon, V. C. (2002a). On merging statecharts and queuing networks to specify and deal with performance models. *Performance 2002 - IFIP WG 7.3 International Symposium on Computer Modeling, Measurement and Evaluation*, Rome, Italy.
- Francês, C. R. L.; Santana, M. J.; R. H. C. Santana, N. L. V.; Solon, V. C. (2002b). Performance evaluation of complex systems: Queuing statecharts approach. *I Workshop em Avaliação de Desempenho de Sistemas Computacionais e de Comunicação (I Wperformance)*, Florianópolis, Santa Catarina, Brasil.
- Francês, C. R. L.; Spolon, R.; Santana, R. H. C.; Santana, M. J. (2000b). *Modelagem e Especificação Utilizando Redes de Petri e Statecharts*. Notas Didáticas do ICMC - USP No 45.
- Fujimoto, R. M. (1999). Parallel and distributed simulation. *Proceedings of the 31st conference on Winter simulation*, p. 122-131. ACM Press.
- Fujimoto, R. M. (2001). Parallel simulation: parallel and distributed simulation systems. *Proceedings of the 33rd conference on Winter simulation*, p. 147-157. IEEE Computer Society.
- Gosling, J.; McGilton, H. (1996). The java language environment. <http://java.sun.com/docs/white/lanenv/index.html>.
- Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, v.8, n.3, p.231-274.

- Intel (2004). Intel performance: Benchmark limitations. http://www.intel.com/performance/resources/benchmark_limitations.htm.
- Jain, R. (1991). *The Art of Computer Systems Performance Analysis: Techniques of Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons.
- Jefferson, D. R. (1985). Virtual time. *ACM Transactions on Programming Languages and Systems*, v.7, n.3, p.404–425.
- Jensen, K.; Huber, P.; Shapiro, R. M. (1990). *Hierarchies in Coloured Petri Nets*.
- Kleinrock, L. (1976). *Queueing Systems*. John Wiley & Sons.
- M., S.; Santana, R. H.; Santana, M. J. (2000). Asda - an automatic distributed simulation environment. *Proceedings of 2000 Summer Computer Simulation Conference (SCSC' 2000)*, Vancouver, Canada.
- Macdougall, M. H. (1987). *Simulating Computing Systems Techniques and Tools*. MIT PRESS.
- Macedo, L. L.; Saad, A. C. C. (1979). *Análise do Desempenho de Computadores: Avaliação, Controle e Otimização*. São Paulo Editora Edgard Blucher.
- Macfadyen, N. (2003). Performance modelling. *BT Technology Journal*, v.21, n.2, p.77–89.
- Maciel, P. R. M.; Lins, R. D.; Cunha, P. R. F. (1996). *Introdução às Redes de Petri e Aplicações*. IC-UNICAMP, Campinas.
- Magalhães, M. N. (1996). *Introdução à Rede de Filas*. ABE - Associação Brasileira de Estatística.
- Meyer, J. F. (1995). Performability evaluation: Where it is and what lies ahead. *Proceedings of the Computer Performance and Dependability Symposium*, p. 334–343, Erlangen, Germany.
- Moore, K. E.; Brennan, J. E. (1995). Alpha/sim simulation software tutorial. *Proceedings of the 1995 Winter Simulation Conference*, p. 387–394.
- Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, n. Volume: 77 Issue: 4, p. 541–580.
- Nonato, S. A. (1999). Utilização de redes de petri para avaliação de sistemas computacionais. Dissertação (Mestrado), Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo.

- Orlandi, R. C. G. S. (1995). Ferramenta para análise de desempenho de sistemas computacionais distribuídos. Dissertação (Mestrado), Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo.
- Pegden, C. D.; Shannon, R. E.; Sadowski, R. P. (1995). *Introduction to Simulation using SIMAN*. 2nd ed., MacGraw-Hill International Editions, New York.
- Petri, C. A. (1966). *Communication with Automata*. Tese (Doutorado), New York. Technical Report RADC-TR-65-377.
- Santana, M. J.; Santana, R. H. C.; Frances, C. R. L.; Orlandi, R. (1997). Tools and methodologies for performance evaluation of distributed systems - a comparison study. *In The Proceedings of the: Summer Computer Simulation Conference*, Arlington, Virginia, United States of the America.
- Santana, R. H. C. (1990). *Performance Evaluation of Lan-Based File Servers*. Tese (Doutorado), University of Southampton.
- Santana, R. H. C.; Santana, M. J.; Orlandi, R. C. G. S.; Spolon, R.; Calônego Júnior, N. (1994). *Técnicas para Avaliação de Desempenho de Sistemas Computacionais*. Notas Didáticas do ICMC - USP No 14 . 31p. 1994.
- Sauer, C. H.; Chandy, K. M. (1981). *Computer Systems Performance Modeling*. Prentice-Hall Series in Advances in Computing Science and Technology.
- Singleton, P. (2002). Performance modelling - what, why, when and how. *BT Technology Journal*, v.20, n.3, p.133-143. .
- Soares, L. F. G. (1990). *Modelagem e Simulação Discreta de Sistemas*. IME-USP, São Paulo.
- Spolon, R. (1994). Um editor gráfico para um ambiente de simulação automático. Dissertação (Mestrado), Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo.
- Suchring, S. (2002). *MySQL A Bíblia*. Campus, Rio de Janeiro, 1a. edição.
- Sun (1996). Api specifications. <http://java.sun.com/j2se/1.4.2/docs/api/>.
- Swain, J. J. (1997). *Simulation on Survey: Tools for process understanding and improvement*.
- Tatsumi, E. S. (2002). *Análise e Avaliação do Uso da Ferramenta ParSMPL em um Ambiente de Simulação Distribuído Automático (ASDA)*. Monografia de Qualificação do Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo.