ASDA - um ambiente de simulação
distribuída automático

Sarita Mazzini Bruschi

ASDA - um ambiente de simulação distribuída automático

Sarita Mazzini Bruschi

Orientadora:

Profa. Dra. ***Regina Helena Carlucci Santana***

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

USP – São Carlos
Fevereiro/2003

(Versão Revisada)

Data da Defesa: 25/11/2002

Visto do Orientador:

*“O pescador sabe que o mar
é perigoso e a tormenta terrível.*

*Mas este conhecimento não
o impede de lançar-se ao mar”*

Vicent Van Gogh

Dedicatória

*À minha família,
pelo apoio e dedicação,
sem os quais, esta conquista
não seria possível.*

Agradecimentos

À Deus por me guiar nesta fase que se encerra e pela constante proteção demonstrada todos os dias em pequenos “sinais”

À minha orientadora, Prof. Dra. Regina Helena Carlucci Santana, pela orientação, pela paciência e principalmente pela incrível capacidade de colocar minhas idéias em ordem.

Ao Prof. Marcos José Santana, pelo apoio e pela importante ajuda no desenvolvimento deste trabalho.

Aos meus pais pelo apoio constante, pela compreensão e por fazerem de tudo para dar uma ótima educação para os filhos. Às minhas irmãs e irmão pelo incentivo, amizade e confiança.

À Regina e ao Marcos, pela amizade e pela confiança depositada em mim. À Cissa, minha irmãzinha são-carlense, pela amizade.

Aos colegas do grupo de Sistemas Distribuídos e Programação Concorrente. No decorrer desses 8 anos em São Carlos, muitos chegaram, muitos já se foram, alguns aqui ainda estão: Adriana, Aletéia, Álvaro, Ana Elisa, Andrezza, Arion, Boca, Célia, Daniel, Douglas, Edmilson, Hermes, João Carlos, Jorge, Kalinka, Luciano, Mara, Márcio, Mário, Naninha, Omar, Paulo Sérgio, Renata, Roberta, Renato, Renato (Japa), Ricardo, Silmara, Simone (Paulo Sérgio), Tatiana, Thais, Thomas e Vera. Obrigada pelas discussões no café, pelas dúvidas solucionadas, pela companhia, pelos churrascos e principalmente pelas festas surpresas.

Às amigas Kalinka, Célia, Naninha, Simone, Tatiana e Andrezza pelos bons e divertidos momentos que passamos juntas. Obrigada pela amizade.

Às amigas do primário Cristiane, Cecília e Arian, pela amizade que já dura 20 anos, apesar da correria e da distância.

Ao Luciano, Márcio e Ricardo pelo valioso auxílio na configuração da rede de computadores e do PVM. Ao Omar pelo auxílio com o MatLab.

À Simone pela paciência na correção das referências bibliográficas e pela companhia na república.

Ao Prof. Edjair Mota pelas inúmeras dúvidas solucionadas em relação à abordagem MRIP e ao Akaroa.

À Prof. Rosângela Penteado pela revisão dos diagramas de UML.

Ao Prof. Marinho pelo auxílio nas questões estatísticas.

À todos os meus professores, com os quais muito aprendi e sem os quais hoje não estaria aqui

Aos funcionários da Seção Técnica de Informática, Sônia, Franz, Dotta, Eduardo e Cabral por estarem sempre dispostos a resolver os problemas da rede e de hardware.

À Beth, Laura e Ana Paula, da Secretaria de Pós-Graduação, por estarem sempre prestativas para o atendimento e na solução de dúvidas

À Adriana e Tatiana, da sessão de convênio, pela atenção dispensada e por estarem sempre dispostas a solucionar todas as dúvidas referentes aos relatórios da FAPESP

À todos os funcionários do ICMC, os quais tornaram nossa jornada possível de ser realizada

Àqueles que, por motivos da correria na finalização, eu tenha esquecido de citar nominalmente

À todos aqueles que contribuíram direta ou indiretamente para a realização deste trabalho

À FAPESP, pelo apoio financeiro

Sumário

Lista de Figuras.....	v
Lista de Tabelas.....	viii
Lista de Siglas.....	ix
Resumo.....	x
Abstract.....	xi
1 Introdução.....	1
2 Simulação	7
2.1 Considerações Iniciais.....	7
2.2 Fases do desenvolvimento de uma simulação	8
2.3 Modelagem.....	11
2.3.1 <i>Statecharts</i>	12
2.3.1.1 Notação.....	13
2.3.1.2 Hierarquia de estados.....	13
2.3.1.3 Concorrência	14
2.3.2 <i>Redes de Petri</i>	15
2.3.2.1 Regras de disparo de Redes de Petri	16
2.3.2.2 Classificação das Redes de Petri	16
2.3.3 <i>Redes de Filas</i>	17
2.3.3.1 Redes de Filas Estendidas	21
2.3.4 <i>Queuing Statecharts</i>	21
2.3.5 <i>Comparação entre as abordagens apresentadas</i>	25
2.4 Análise de Saída	25
2.4.1 <i>Tipos de Simulação</i>	26
2.4.2 <i>Geração de números aleatórios</i>	27
2.4.3 <i>Medidas utilizadas</i>	28
2.4.4 <i>Período Transiente ou Período de Warm-Up</i>	29
2.4.5 <i>Correlação entre as observações</i>	29
2.4.6 <i>Métodos para Análise de Saída</i>	30
2.4.6.1 Método da Replicação/Deleção	31
2.4.6.2 Método da Média dos Lotes (BM – Batch Means)	32
2.4.6.3 Método da Padronização de Séries Temporais (STS – Standardized Time Series).....	33
2.4.6.4 Análise Espectral (SA – Spectral Analysis)	34
2.4.7 <i>Comparação entre os métodos descritos</i>	35
2.5 Considerações Finais.....	36
3 Simulação Discreta Distribuída / Paralela	37
3.1 Considerações Iniciais.....	37
3.2 <i>Single Replication in Parallel - SRIP</i>	39
3.2.1 <i>Protocolos para SRIP</i>	40
3.2.2 <i>Protocolos Conservativos</i>	41
3.2.2.1 CMB (Prevenção do Deadlock através de Mensagens Nulas)	41
3.2.2.2 Transmissão de Mensagens Nulas sob Demanda	43

3.2.2.3	Detecção e Recuperação do Deadlock	45
3.2.3	<i>Protocolos Otimistas</i>	45
3.2.3.1	Time Warp.....	46
3.2.4	<i>Comparação entre os protocolos conservativos e otimistas</i>	49
3.3	<i>Mutiple Replications in Parallel – MRIP</i>	51
3.4	Comparação entre <i>SRIP</i> e <i>MRIP</i>	53
3.5	Fases de desenvolvimento para uma simulação distribuída	55
3.6	Recursos disponíveis para o desenvolvimento de simulações distribuídas	56
3.6.1	<i>Linguagens e Bibliotecas</i>	56
3.6.2	<i>Ambientes de Simulação Automáticos</i>	59
3.6.2.1	ARENA.....	60
3.6.2.2	OMNeT++	60
3.6.2.3	ASiA	61
3.6.2.4	UCLA Simulation Environment	63
3.6.2.5	DISplay: A System for visual-interaction in distributed simulation	64
3.6.2.6	Akaroa-2.....	64
3.7	Considerações Finais.....	65
4	ASDA – Um Ambiente de Simulação Distribuída Automático	67
4.1	Considerações Iniciais.....	67
4.2	ASDA	68
4.3	Módulo de Interface com o usuário.....	72
4.3.1	<i>Interface com o Usuário</i>	72
4.3.2	<i>Descrição das atividades do Módulo de Interface</i>	73
4.3.3	<i>Especificação do Módulo de Interface em UML</i>	81
4.4	Módulo Avaliador.....	87
4.4.1	<i>Nível 1</i>	88
4.4.2	<i>Nível 2</i>	89
4.4.3	<i>Nível 3</i>	92
4.4.4	<i>Utilização de Inteligência Artificial no Módulo Avaliador</i>	98
4.4.5	<i>Especificação do Módulo Avaliador em UML</i>	103
4.5	Módulo Gerador	104
4.5.1	<i>Especificação do Módulo Gerador em UML</i>	110
4.6	Módulo de Interface de Software.....	114
4.7	Módulo Executor	116
4.8	Módulo Replicador	116
4.9	Módulo Escalonamento	118
4.10	Diagramas UML.....	121
4.10.1	<i>Diagrama de Classes do ASDA</i>	121
4.10.2	<i>Diagrama de Atividades do ASDA</i>	124
4.11	Comparação entre os ambientes descritos.....	125
4.12	Considerações Finais.....	125
5	Implementação do Protótipo do ASDA.....	127
5.1	Considerações Iniciais.....	127
5.2	Módulo de Interface com o Usuário	129
5.2.1	<i>A escolha da ferramenta</i>	129

5.2.2	<i>A ferramenta Kylix</i>	130
5.2.3	<i>A Interface Gráfica do ASDA</i>	132
5.2.4	<i>Base de Dados do ASDA</i>	134
5.3	Módulo Replicador	136
5.3.1	<i>ASDA-MRIP</i>	137
5.3.2	<i>Submódulo Mestre</i>	139
5.3.3	<i>Submódulo Analisador Global</i>	140
5.3.4	<i>Submódulo Escravo</i>	140
5.3.4.1	Período transiente.....	140
5.3.4.2	Período de cálculo das estimativas	141
5.3.5	<i>Submódulo Comunicação</i>	142
5.3.6	<i>Submódulo Debug</i>	144
5.3.7	<i>Estrutura de subdiretórios e execução da simulação</i>	147
5.4	Módulo Interface de Software	149
5.5	Considerações Finais.....	153
6	Testes realizados com o ASDA	155
6.1	Considerações Iniciais	155
6.2	Metodologia adotada	155
6.3	Características da arquitetura.....	159
6.4	Validação do ASDA-MRIP.....	160
6.4.1	<i>Sistema M/M/1</i>	161
6.4.2	<i>Sistema Computacional Distribuído</i>	164
6.5	Análise dos resultados obtidos	168
6.5.1	<i>Influência do número de replicações no tempo de execução da simulação</i>	169
6.5.1.1	Granulação do processo.....	170
6.5.1.2	Método de análise e característica da arquitetura	177
6.5.2	<i>Conclusões sobre a análise dos resultados obtidos</i>	179
6.6	Considerações Finais.....	180
7	Conclusões	181
7.1	Conclusões Gerais	181
7.2	Contribuições desta Tese.....	183
7.3	Sugestões para Trabalhos Futuros.....	185
	Referências Bibliográficas.....	187
	Apêndice A. Computação Paralela / Distribuída	197
	A.1. <i>Ambientes de Passagem de Mensagem</i>	199
	A.2. <i>PVM – Parallel Virtual Machine</i>	199
	A.2.1. <i>O modelo computacional do PVM</i>	200
	A.2.2. <i>Comunicação e Sincronismo no PVM</i>	201
	Apêndice B. UML (Unified Modelling Language).....	203
	B.1 <i>Blocos de construção da UML</i>	204
	B.1.1 <i>Itens</i>	204

B.1.1.1 Itens Estruturais	205
B.1.1.2. Itens Comportamentais	207
B.1.1.3 Itens Agrupamento	208
B.1.1.4 Itens Anotacionais	208
<i>B.1.2 Relacionamentos</i>	<i>208</i>
B.1.2.1 Dependência	209
B.1.2.2 Associação	209
B.1.2.3. Generalização.....	209
B.1.2.4. Realização	210
<i>B.1.3 Diagramas.....</i>	<i>210</i>
B.1.3.1 Diagramas de classes e objetos.....	211
B.1.3.2. Diagramas de casos de uso.....	211
B.1.3.3. Diagramas de seqüências	212
B.1.3.4. Diagrama de colaborações.....	212
B.1.3.5. Diagramas de estados.....	213
B.1.3.6. Diagramas de atividades.....	214
B.1.3.7. Diagramas de componentes.....	214
B.1.3.8. Diagramas de implantação.....	215
Apêndice C. Códigos dos Programas de Simulação	217

Lista de Figuras

Figura 2.1. Etapas para o desenvolvimento de uma simulação.....	8
Figura 2.2. Decomposição OR.....	14
Figura 2.3. Decomposição AND.....	14
Figura 2.4. Elementos básicos de uma Rede de Petri (a) <i>token</i> , (b) lugar, (c) transição, (d) arco.....	15
Figura 2.5. Modelo de uma Rede de Petri com os elementos básicos.....	16
Figura 2.6. Centro de Serviço: uma fila e um servidor.....	18
Figura 2.7. Exemplos de centros de serviço.....	18
Figura 2.8. Redes de filas abertas, fechadas e mistas.....	19
Figura 2.9. Representação de um modelo utilizando centro passivo.....	21
Figura 2.10 Template para Fonte.....	22
Figura 2.11. Template para Fila.....	22
Figura 2.12. Template para Servidor.....	23
Figura 2.13. Template para Sorvedouro.....	23
Figura 2.14. Template para Centro de Serviço.....	23
Figura 2.15. Servidor de arquivos representado utilizando Redes de Filas.....	24
Figura 2.16. Servidor de arquivos representado utilizando <i>Queueing Statecharts</i>	24
Figura 3.1. Relação entre as classes de aplicação de simulação.....	37
Figura 3.2. Ocorrência de erros de causa e efeito.....	40
Figura 3.3. <i>LVT</i> de um processo lógico.....	42
Figura 3.4. Situação de <i>Deadlock</i>	42
Figura 3.5. Transmissão de Mensagens Nulas sob Demanda.....	44
Figura 3.6. <i>Deadlock</i> no pedido de avanço de relógio.....	44
Figura 3.7. Estrutura do processo lógico A no Time Warp.....	47
Figura 3.8. Chegada de uma mensagem <i>straggler</i>	47
Figura 3.9. Interação entre fatores que influenciam o desempenho dos protocolos de sincronização.....	50
Figura 3.10. Utilização da abordagem <i>MRIP</i>	54
Figura 3.11. Utilização da abordagem <i>SRIP</i>	54
Figura 3.12. Fases de desenvolvimento de uma simulação distribuída.....	55
Figura 3.13. Simulação de uma rede <i>token ring</i> no OMNeT++.....	61
Figura 3.14. Visão geral do ASiA.....	62
Figura 3.15. Tela principal do ASiA com um modelo de redes de filas.....	62
Figura 3.16. Visão geral do UCLA Simulation Environment.....	64
Figura 3.17. Arquitetura do Akaroa.....	65
Figura 4.1. Diagrama Modular do ASDA.....	70
Figura 4.2. Exemplo de um registro do arquivo de usuários.....	76
Figura 4.3. Representação de um modelo de Rede de Filas através de Grafos.....	77
Figura 4.4. Diagrama de Casos de Uso do ASDA.....	81
Figura 4.5. Escolha entre as abordagens SRIP e MRIP segundo a plataforma (processamento x comunicação).....	90
Figura 4.6. Escolha entre as abordagens SRIP e MRIP segundo a duração da simulação e a duração do <i>warm-up</i>	91
Figura 4.7. Escolha entre as abordagens SRIP e MRIP segundo o tamanho do modelo e o número de processadores.....	92

Figura 4.8. Relação entre o tamanho do modelo, a precisão requerida e o tamanho da simulação	93
Figura 4.9. Partição do modelo em processos lógicos analisando a realimentação	94
Figura 4.10. Partição do modelo em processos lógicos analisando	95
Figura 4.11. Escolha entre os protocolos CMB, Time Warp e CMB $\Rightarrow$ Time Warp.....	96
Figura 4.12. Estrutura básica de um sistema classificador	98
Figura 4.13. Gráfico gerado pelo <i>toolbox</i> de lógica fuzzy do software Matlab para a escolha entre as abordagens MRIP e SRIP	101
Figura 4.14. Função de pertinência para a variável comunicação	102
Figura 4.15. Função de pertinência para a variável processamento	102
Figura 4.16. Estrutura de arquivos no módulo Gerador	105
Figura 4.17. Arquivo gabarito.dat para a extensão funcional SMPL	106
Figura 4.18. Gabarito para geração de código MRIP	107
Figura 4.19. Gabarito para geração de código SRIP, utilizando a extensão ParSMPL	109
Figura 4.20. Relação entre os processos mestre e escravo na simulação MRIP	117
Figura 4.21. Dois estágios de cada replicação	118
Figura 4.22. Interação PVM e AMIGO	120
Figura 4.23. Diagrama de classes do ASDA, visão dos módulos.....	121
Figura 4.24. Diagrama de classes do ASDA, módulo Interface e Especificação Gráfica	122
Figura 4.25. Diagrama de Classes do ASDA, módulo Geração dos Programas de Simulação	123
Figura 4.26. Diagrama de Atividades do ASDA	124
Figura 5.1. Módulos do Kylix e Delphi	131
Figura 5.2. Tela de identificação do usuário.....	133
Figura 5.3. Tela inicial de entrada do ASDA para (a) super-usuário (b) usuário de simulação	133
Figura 5.4. Tela de início para (a) super-usuário (b) usuário de simulação	134
Figura 5.5. Tela de Especificação Gráfica do ASDA	135
Figura 5.6. Tabela Usuários definida para o ASDA	136
Figura 5.7. Primeira estrutura idealizada para um ambiente de simulação MRIP	138
Figura 5.8. Estrutura do ambiente para simulação MRIP	138
Figura 5.9. Estrutura detalhada da ASDA-MRIP	139
Figura 5.10. Exemplo da heurística no. 5 para detecção do período transiente	141
Figura 5.11. Código da função <i>start_communication</i>	144
Figura 5.12. Comunicação entre o processo mestre e os escravos.....	145
Figura 5.13. Estrutura de subdiretórios do ASDA-MRIP	147
Figura 5.14. Trechos a serem inseridos no código do programa de simulação	150
Figura 6.1. Modelo de um sistema M/M/1.....	161
Figura 6.2. Intervalo de confiança das execuções do modelo M/M/1.....	164
Figura 6.3. Modelo de um sistema computacional distribuído.....	165
Figura 6.4. Intervalo de confiança das execuções do modelo sistema distribuído	168
Figura 6.5. Intervalos de confiança dos tempos de simulação do modelo M/M/1 utilizando o método Análise Espectral.....	173
Figura 6.6. Intervalos de confiança dos tempos de simulação do modelo Sist. Distribuído utilizando o método Análise Espectral.....	174
Figura 6.7. Intervalos de confiança dos tempos de simulação do modelo Sist. Distribuído com repetição utilizando o método Análise Espectral.....	174
Figura 6.8. Intervalos de confiança dos tempos de simulação do modelo M/M/1 utilizando o método Padronização de Séries Temporais	174

Figura 6.9. Intervalos de confiança dos tempos de simulação do modelo Sist. Distribuído utilizando o método Padronização de Séries Temporais	175
Figura 6.10. Intervalos de confiança dos tempos de simulação do modelo Sist. Distribuído com repetição utilizando o método Padronização de Séries Temporais	175
Figura 6.11. Intervalos de confiança dos tempos de execução da simulação do modelo M/M/1 utilizando os métodos Análise Espectral e Padronização de Séries Temporais.....	177
Figura 6.12. Intervalos de confiança dos tempos de execução da simulação do modelo sistema distribuído utilizando os métodos Análise Espectral e Padronização de Séries Temporais	178
Figura 6.13. Intervalos de confiança dos tempos de execução da simulação do modelo sistema distribuído com repetição utilizando os métodos Análise Espectral e Padronização de Séries Temporais.....	178
Figura A.1. Estrutura de um multiprocessador com memória compartilhada	198
Figura A.2. Estrutura de um multicomputador com memória distribuída.....	198
Figura B.1. Modelagem da arquitetura de um sistema	203
Figura B.2. Representação gráfica de uma classe	205
Figura B.3. Representação gráfica de uma interface	205
Figura B.4. Representação gráfica de uma colaboração.....	206
Figura B.5. Representação gráfica de um caso de uso	206
Figura B.6. Representação gráfica de uma classe ativa	206
Figura B.7. Representação gráfica de um componente.....	207
Figura B.8. Representação gráfica de um nó.....	207
Figura B.9. Representação gráfica de uma mensagem	207
Figura B.10. Representação gráfica de um estado	208
Figura B.11. Representação gráfica de um pacote	208
Figura B.12. Representação gráfica de uma nota	208
Figura B.13. Dependência	209
Figura B.14. Associação	209
Figura B.15. Generalização	209
Figura B.16. Realização.....	210
Figura B.17. Diagrama de classes.....	211
Figura B.18. Diagrama de Casos de Uso.....	212
Figura B.19. Diagrama de Seqüência	213
Figura B.20. Diagrama de Colaboração	213
Figura B.21. Diagrama de Estados.....	214
Figura B.22. Diagrama de Atividades	214
Figura B.23. Diagrama de Componentes	215
Figura B.24. Diagrama de Implantação	215
Figura C.1. Código MRIP gerado a partir de um programa de simulação seqüencial utilizando a biblioteca SMPL.	218
Figura C.2. Código MRIP gerado a partir de um programa de simulação seqüencial utilizando a biblioteca SimPack.	220
Figura C.3. Código MRIP gerado a partir de um programa de simulação seqüencial utilizando a biblioteca SSS.....	222
Figura C.4. Código do modelo M/M/1.....	223
Figura C.5. Código do modelo do sistema distribuído	225
Figura C.6. Código do modelo do sistema distribuído com repetição	227

Lista de Tabelas

Tabela 2.1. Eventos-padrão do <i>Queueing Statecharts</i> e suas semânticas	23
Tabela 3.1 Comparação entre os protocolos conservativo e otimista (Fujimoto, 2000).....	50
Tabela 3.2. Comparação entre diversas linguagens para SRIP (Low et al., 1999)	57
Evento	57
Tabela 3.3. Comparação entre diversas bibliotecas para SRIP	58
Sequencial.....	58
Tabela 4.1. Associação entre a representação gráfica utilizando Redes de Filas, <i>Queueing Statecharts</i> e o grafo a ser armazenado	78
Tabela 4.2. Informações a serem armazenadas em cada nó do modelo	79
Tabela 4.3. Associação entre as características e a classe para o nível 2 do módulo Avaliador	100
Tabela 4.4. Associação entre as características e a classe para o nível 3 do módulo Avaliador	100
Tabela 5.1. Relação das funções que fazem parte do arquivo <i>communication.c</i>	143
Tabela 5.2. Relação dos arquivos do ASDA-MRIP e suas funções	148
Tabela 5.3. Relação das funções de funções de probabilidade nas bibliotecas ASDA-MRIP, SMPLX, SimPack e SSS.....	152
Tabela 6.1. Configuração dos testes realizados para os modelo de validação.....	156
Tabela 6.2. Configuração dos testes realizados para os modelo de análise	158
Tabela 6.3. Configuração das máquinas utilizadas nos experimentos	159
Tabela 6.4. Resultados obtidos nas execuções do sistema M/M/1.....	163
Tabela 6.5. Resultados do sistema M/M/1 - validação do nível de confiança e precisão.....	162
Tabela 6.6. Resultados obtidos nas execuções do sistema distribuído	167
Tabela 6.7. Resultados do sistema distribuído - validação do nível de confiança e precisão	166
Tabela 6.8. Tempo de simulação do modelo M/M/1 e método Análise Espectral.....	171
Tabela 6.9. Tempo de simulação do modelo sistema distribuído e método Análise Espectral	171
Tabela 6.10. Tempo de simulação do modelo sistema distribuído com repetição e método Análise Espectral.....	171
Tabela 6.11. Tempo de simulação do modelo M/M/1 e método Padronização de Séries Temporais .	172
Tabela 6.12. Tempo de simulação do modelo sistema distribuído e método Padronização de Séries Temporais.....	172
Tabela 6.13. Tempo de simulação do modelo sistema distribuído com repetição e método Padronização de Séries Temporais	173
Tabela 6.14. Variação do tempo de execução em relação ao tempo de execução com uma replicação	176

Lista de Siglas

ALSP – Aggregate Level Simulation Protocol
BM – Batch Means
CLX – Component Library Cross Plataform
CMB – Chandy, Misra e Bryant
CMR – Combined Multiple Recursive
DIDO – Deterministic In, Deterministic Out
DIS – Distributed Interactive Simulation
FCFS – First Come, First Served
FIFO – First In, First Out
GVT – Global Virtual Time
HLA – High Level Architecture
IDE – Integrated Development Environment
LCG – Linear Congruential Generator
LVT – Local Virtual Time
MEF – Máquina de Estado Finito
MIMD – Multiple Instruction Multiple Data
MPI – Message Passing Interface
MRIP – Multiple Replication in Parallel
MSE – Mean Square Error
OBM – Overlapping Batch Means
PADS – Parallel and Distributed Simulation
PDES – Parallel Discrete Event Simulation
PL – Processo Lógico
PVM – Parallel Virtual Machine
RAD – Rapid Application Development
RIRO – Randon In, Random Out
RR – Round Robin
SA – Spectral Analysis
SRIP – Single Replication in Parallel
STS – Standardized Time Series
UML – Unified Modelling Language
VCL – Visual Components Library
VLSI – Very Large-Scale Integration
WYSIWYG - What You See Is What You Get

Resumo

Esta tese propõe um ambiente automático para desenvolvimento de simulação distribuída – ASDA (**A**mbiente de **S**imulação **D**istribuída **A**utomático), que tem como objetivo principal facilitar a utilização e desenvolvimento de simulação distribuída. As funcionalidades definidas no ASDA tornam-o diferente de todos os outros ambientes encontrados na literatura. A especificação do ASDA foi realizada através de um diagrama modular composto por sete módulos e também com o auxílio da UML (*Unified Modelling Language*), através da utilização de três de seus diagramas: de casos de uso, de classes e de atividades.

O ASDA permite aos usuários a utilização de simulação distribuída através da definição de uma nova simulação ou da replicação de um programa de simulação já desenvolvido. Se a opção for pelo desenvolvimento de um novo programa de simulação, o usuário deve fornecer o modelo e os parâmetros e o ambiente se encarrega de gerar o código do programa de simulação utilizando a abordagem que proporciona o melhor desempenho, levando em consideração as características do modelo e da plataforma.

Além da especificação do ASDA, esta tese definiu um protótipo do ambiente com o objetivo de mostrar sua viabilidade de utilização. Neste protótipo, três módulos foram implementados, destacando-se o módulo Replicador, que utiliza a abordagem MRIP (*Multiple Replication in Parallel*).

Esta tese contribui também com a definição de algumas diretrizes para a utilização da abordagem MRIP. A base para essa definição foram os resultados obtidos com a utilização do módulo Replicador.

Abstract

This thesis proposes an automatic environment for the development of distributed simulation – ASDA (Ambiente de Simulação Distribuída Automático – in Portuguese), whose main goal is to make easier the use and development of distributed simulation. The ASDA functionality makes it different from all other environments found in the literature. The ASDA has been specified through a modular diagram, composed of seven modules built with the help of the UML (Unified Modelling Language), using three of its diagrams: use case, class and activity.

ASDA users can define the distributed simulation by means of the specification of a new simulation program or the replication of a simulation program already developed. If the user chooses to develop a new simulation program, he must only provide the model and the parameters. The environment will then generate the simulation program code using the approach that provides the best performance considering the model and platform characteristics.

Besides the specification, this thesis presents a prototype of the ASDA environment with the goal of showing its viability. Three modules have been implemented for the prototype, highlighting the Replication module, which uses the MRIP (Multiple Replication in Parallel) approach.

Another contribution of this thesis is the definition of a set of guidelines to the utilization of the MRIP approach. The basis to define these procedures was the results obtained with the utilization of the Replication module.

Introdução

Nunca a humanidade evoluiu tanto como no século passado (séc. XX). Essa evolução deu-se principalmente devido ao avanço tecnológico e da ciência, que trouxe incontáveis benefícios, mas ao mesmo tempo, preocupações. Preocupações com relação a investimento financeiro, tempo despendido no desenvolvimento de novos sistemas (computacionais ou não), benefícios da substituição de um sistema antigo por um novo, etc.

Em termos computacionais, essa evolução aconteceu principalmente nas últimas quatro décadas e uma das áreas que se tem destacado devido principalmente às preocupações citadas no parágrafo anterior é a avaliação de desempenho. A avaliação de desempenho é muito importante quando se tem por objetivo minimizar ou eliminar algumas dessas preocupações. Técnicas para avaliação de desempenho podem ser aplicadas tanto à sistemas existentes quanto a sistemas não existentes, tanto para avaliação de um único sistema quanto para a comparação entre sistemas, tanto para uma avaliação de baixo nível quanto para avaliação de alto nível, etc.

As diversas técnicas para avaliação de desempenho que vêm sendo propostas baseiam-se em duas abordagens: as técnicas de aferição, onde as informações requeridas por um estudo podem ser obtidas a partir do próprio sistema, e as técnicas de modelagem onde se constrói um modelo representativo do sistema. A escolha entre as abordagens depende do objetivo da avaliação, do sistema a ser avaliado e do nível de detalhe a ser considerado. Nas técnicas de aferição é necessário que o sistema a ser avaliado esteja pronto para que as informações possam ser coletadas. Um exemplo das técnicas de aferição são os *benchmarks*, utilizados principalmente na comparação de sistemas existentes.

As técnicas de modelagem baseiam-se na construção de um modelo que represente as características a serem analisadas no sistema real. A solução desses modelos pode ser obtida através de soluções analíticas ou por simulação. As soluções analíticas são atrativas devido a precisão dos resultados obtidos. No entanto, a falta de flexibilidade e a dificuldade em representar características essenciais de determinados sistemas computacionais tornam a utilização dessa técnica inviável na maioria dos casos de avaliação de sistemas computacionais.

A simulação é uma outra forma de solucionar um modelo. Neste caso, o sistema é abstraído em um modelo e a partir deste modelo um programa de simulação é gerado. Esta técnica oferece grande flexibilidade e baixo custo pois qualquer alteração no sistema é refletida com maior facilidade no programa de simulação. Assim, a adoção de simulação é atrativa tanto para sistemas existentes como para verificação de projetos. No entanto, os

resultados obtidos com a simulação não são muito precisos e dependem de análises estatísticas.

Uma simulação pode ser classificada como determinística ou estocástica, de acordo com os dados fornecidos na entrada. A maioria dos sistemas a serem simulados são transformados em simulações estocásticas, onde os dados de entrada são gerados aleatoriamente a partir de uma distribuição de probabilidade. Devido a aleatoriedade dos dados de entrada, a simulação precisa ser repetida várias vezes de modo a garantir que as variáveis estimadas minimizem a influência das variáveis aleatórias. É aqui que a simulação apresenta uma grande desvantagem: o tempo de execução da simulação.

Com o objetivo de diminuir o tempo de processamento de uma simulação, duas abordagens foram propostas e têm sido estudadas: simulação distribuída ou *SRIP* (*Single Replication in Parallel*) (Jefferson, 1985; Misra, 1986; Fujimoto, 1990) e Múltiplas Replicações em Paralelo (*MRIP – Multiple Replication in Parallel*) (Heidelberger, 1988; Ewing *et al.*, 1999). Tanto uma como a outra abordagem fazem uso dos benefícios de uma outra área que vem apresentando grandes avanços: a Computação Paralela. O objetivo principal da Computação Paralela é aumentar o desempenho, diminuindo o tempo para se obter um resultado (Almasi & Gottlieb, 1994). O que diferencia as abordagens *SRIP* e *MRIP* é o modo como o modelo será distribuído nos vários processadores disponíveis para a simulação.

Na abordagem *SRIP*, o sistema é particionado em processos lógicos e cada processo lógico é mapeado em um processador. Estes processos se comunicam através da troca de mensagens. Para garantir que os eventos sejam simulados na devida ordem, diversos protocolos de sincronização foram desenvolvidos. O objetivo dos protocolos é sincronizar os tempos da simulação uma vez que a evolução de cada processo lógico depende dos eventos que podem ocorrer em outros processos lógicos.

Na abordagem *MRIP* o sistema não é particionado. Replicações independentes de um mesmo programa de simulação sequencial são executadas em paralelo. Cada replicação envia seus resultados (variáveis estimadas) para um analisador global, onde as médias finais são calculadas. Quando a precisão requerida é atingida, a simulação é encerrada.

Desenvolver uma simulação sequencial não é uma tarefa muito fácil, principalmente para usuários com pouca experiência. Necessita-se conhecimentos em modelagem, programação, linguagens para simulação e análise dos resultados (probabilidade e estatística). O conceito de Ambiente de Simulação Automático foi proposto com o objetivo de auxiliar o desenvolvimento de um programa de simulação, afastando o usuário da tarefa de transcrição do modelo em um programa de simulação e também da análise estatística dos resultados. A maioria dos Ambientes de Simulação desenvolvidos até o momento é voltada à abordagem sequencial.

Se desenvolver uma simulação sequencial confiável é uma tarefa difícil, adotar uma das

abordagens para paralelização da simulação é uma tarefa mais árdua ainda. Além dos conhecimentos citados, necessita-se também de conhecimentos ligados a Computação Paralela (início e finalização dos processos concorrentes, comunicação e sincronismo).

Tendo a consciência de que o desenvolvimento de uma simulação distribuída necessita de um conjunto de conhecimento que nem sempre é simples de ser adquirido, este trabalho de doutorado apresenta a especificação e a implementação de um protótipo de um ambiente para desenvolvimento automático de Simulação Distribuída.

O ambiente, denominado ASDA – Ambiente de Simulação Distribuída Automático, tem por objetivo principal fornecer aos usuários de simulação distribuída um ambiente onde a simulação possa ser desenvolvida sem a necessidade de muitos conhecimentos e experiências nas áreas de Computação Paralela, Simulação ou Simulação Distribuída. Neste caso, o usuário deve ter conhecimento do sistema a ser simulado, de como desenvolver o modelo desse sistema e dos objetivos da simulação, ou seja, quais os resultados que a simulação deve fornecer.

O objetivo do ASDA é oferecer ao usuário um ambiente flexível e amigável. Flexível no sentido de oferecer diversas formas de utilização e de disponibilizar diferentes abordagens de simulação. Para usuário pouco experientes, o ASDA oferece a facilidade de geração automática do programa de simulação, escolha automática da abordagem a ser adotada dentre outras características. Por outro lado, para usuários mais experientes o ASDA oferece a facilidade de utilização de um código próprio, de modificação do código gerado e definição da abordagem a ser utilizada.

O ASDA pode ser considerado um ambiente amigável por oferecer ao usuário a possibilidade de representar seu modelo, parametrizar esse modelo e definir todas as características desejadas para a simulação através de uma interface gráfica de fácil interação.

Assim, no ASDA o usuário tem a possibilidade de escolher entre desenvolver uma nova simulação seqüencial ou paralela (utilizando as abordagens *SRIP*, *MRIP* ou *SRIP + MRIP*) ou utilizar um programa de simulação já desenvolvido e replicá-lo. De forma a fornecer ao usuário todas essas possibilidades, o ASDA foi especificado em 7 módulos: Interface com o Usuário, Avaliador, Gerador, Executor, Interface de Software, Replicador e Escalonamento.

Toda a interação do usuário com o ambiente é fornecida pelo módulo de Interface com o Usuário. É através desse módulo que o usuário pode definir o modelo utilizando as técnicas de modelagem Redes de Filas ou *Queueing Statecharts*. Definido o modelo, o programa de simulação é gerado com base nas linguagens/bibliotecas de simulação disponíveis no ambiente. A abordagem a ser adotada para a geração do programa de simulação pode ser escolhida pelo usuário ou pelo próprio ambiente, através do módulo Avaliador. Este módulo tem como função auxiliar o usuário nas várias decisões que devem ser tomadas durante o processo de definição do modelo e geração do código de simulação. As informações necessárias para a definição do módulo Avaliador foram coletadas a partir de trabalhos que

comparam as abordagens SRIP e MRIP e trabalhos sobre o desempenho dos protocolos para SRIP.

Além de poder desenvolver um novo programa de simulação, o ASDA fornece ao usuário a possibilidade de utilizar simulação distribuída com um programa de simulação já desenvolvido. Para isso, o módulo Interface de Software fornece interfaces para diversas linguagens/bibliotecas de simulação de modo a permitir a replicação do programa.

Com a definição do ambiente, três características do ASDA se destacam: a possibilidade de desenvolvimento de programas de simulação distribuída utilizando várias abordagens (SRIP, MRIP ou SRIP + MRIP), a possibilidade de utilizar um programa de simulação já desenvolvido e replicá-lo e ainda o auxílio fornecido para decisões referentes a escolha da abordagem e escolha do protocolo de sincronização. Dentre os ambientes estudados, nenhum possui todas essas características.

De forma a mostrar a viabilidade de utilização do ambiente proposto, foi desenvolvido um protótipo que conta com a implementação de 3 módulos. Dentre os módulos implementados, destaca-se o módulo Replicador onde sua implementação contribuiu para que algumas conclusões sobre a utilização da abordagem MRIP pudessem ser obtidas.

De forma a relatar o desenvolvimento deste trabalho, a tese está dividida em sete capítulos: um capítulo de Introdução, dois capítulos de revisão bibliográfica, dois capítulos com a definição e o protótipo do ambiente proposto, um capítulo com os resultados e conclusões dos testes realizados com o ambiente e um capítulo com as conclusões. Além disso, três apêndices são utilizados: dois para definir conceitos utilizados no decorrer da tese e um com o código dos programas ilustrados nesta tese.

Para a especificação do ambiente, a princípio foram estudados os conceitos referentes à simulação e computação paralela, os quais se unem para a definição de simulação distribuída. O capítulo 2 discute com detalhes aspectos da utilização de simulação como técnica de avaliação de desempenho. Dentre os tópicos revisados salienta-se as fases para o desenvolvimento de uma simulação, destacando-se duas fases: a modelagem de sistemas (através de redes de filas, redes de petri, *statecharts* e *queueing statecharts*) e os métodos para análise dos resultados da simulação.

No capítulo 3 são apresentadas as abordagens para Simulação Distribuída: *SRIP* e *MRIP*. Uma comparação entre essas abordagens salientando as vantagens e desvantagens de cada é apresentada. Além das definições e comparações, este capítulo apresenta algumas linguagens, bibliotecas e ambientes automáticos para o desenvolvimento de simulações *SRIP* e *MRIP*.

Com os conceitos de simulação, computação paralela e simulação distribuída solidificados e sobretudo, após um estudo das necessidades de um ambiente de simulação distribuída automático e dos ambientes disponíveis para este fim, passou-se à especificação do

ASDA.

No capítulo 4 o Ambiente de Simulação Distribuída Automático (ASDA) é especificado em detalhes. A especificação é feita informalmente através de um diagrama modular e cada um dos módulos é definido com detalhes. Além da definição informal, este capítulo apresenta uma especificação do ASDA baseada na linguagem de modelagem UML através de três diagramas: de Casos de Uso, de Classes e de Atividade.

O capítulo 5 apresenta o desenvolvimento do protótipo do ambiente proposto. Neste protótipo, foram implementados os módulos Interface com o Usuário, Interface de Software e Replicador. O módulo Interface de Software está sendo implementado com o auxílio de um projeto de mestrado, utilizando o ambiente Kylix. Os módulos Interface de Software e Replicador foram implementados utilizando a linguagem C e o sistema operacional Linux.

No capítulo 6 estão apresentados os resultados dos testes realizados com a implementação do módulo Replicador. Este capítulo apresenta a validação do módulo através dos resultados da simulação de dois modelos: o de um sistema M/M/1 e o de um sistema computacional distribuído. Além da validação, este capítulo ainda contribui com algumas conclusões sobre a utilização da abordagem MRIP.

O capítulo 7 encerra o trabalho, apresentando uma conclusão final, as contribuições relevantes deste trabalho de doutorado, as sugestões para trabalhos futuros e a conseqüente continuidade do trabalho.

No apêndice A os conceitos referentes a Computação Paralela e os ambientes de passagem de mensagens são discutidos, destacando-se os ambientes PVM e MPI. No apêndice B a linguagem de modelagem UML é discutida, apresentando seus principais componentes e diagramas. O apêndice C contém os códigos dos programas de simulação utilizados.

Simulação

2.1 Considerações Iniciais

Atualmente os sistemas computacionais estão sendo projetados de modo a permitirem configurações com muitos e cada vez melhores componentes de hardware. Exemplos disso são as arquiteturas com vários processadores e as redes que conectam centenas de computadores. Os custos para projeto e desenvolvimento desses sistemas podem ser significativamente reduzidos se houver uma técnica para a análise de desempenho que possa prever o comportamento futuro de um projeto ou mesmo de algumas alterações em um sistema existente (Bagrodia *et al.*, 1998).

A simulação é atualmente uma das ferramentas disponíveis para análise de desempenho de sistemas computacionais e se tornou atrativa e atuante nas mais diversas áreas de aplicação devido à sua versatilidade (pode ser usada em diferentes ocasiões), flexibilidade (é adaptável a novas e diferentes situações) e baixo custo (com um mesmo programa pode-se simular diferentes situações) (Pegden *et al.*, 1995).

Como exemplo da utilização de simulação, pode-se citar: aplicações militares (simulações de guerra e ambientes de treinamento), projetos aeroespaciais, sistemas urbanos, meteorologia, projetos de engenharia, sistemas de manufatura, sistemas ecológicos e ambientais e sistemas de computação (hardware, redes de comunicação, arquiteturas, etc.) (Pegden *et al.*, 1995; Banks, 1998; Meng, 1999; Fujimoto, 2000).

O processo de desenvolvimento de uma simulação envolve diversas etapas. Primeiramente é necessário especificar o modelo, abstraindo as características mais importantes do sistema. Tendo o sistema sido modelado, é necessário transformar o modelo em um programa de simulação. Em simulações estocásticas, devido à aleatoriedade dos dados de entrada, o programa deve ser executado diversas vezes a fim de garantir que a influência da aleatoriedade nos resultados finais seja minimizada.

De modo a tornar mais clara essa sequência, na próxima seção são discutidos alguns conceitos básicos referentes à simulação de sistemas. As formas de representação de modelos e a análise dos resultados são apresentados em maiores detalhes devido a sua importância para este trabalho.

2.2 Fases do desenvolvimento de uma simulação

Simular é o processo de desenvolver um modelo matemático ou lógico de um sistema real e então realizar experimentos de modo a prever o comportamento real do sistema (Meng, 1999). Para realizar uma simulação algumas etapas devem ser observadas (figura 2.1), as quais são descritas a seguir (Banks, 1998).

Figura 2.1. Etapas para o desenvolvimento de uma simulação

Formulação do problema

Para descrever o sistema deve-se ter um bom entendimento deste para que se possa apresentá-lo sem dubiedade, não negligenciando características relevantes nem considerando detalhes desnecessários que podem resultar em futuros erros.

Ajuste dos objetivos e planos

É nesta fase que os objetivos a serem alcançados são traçados. Os objetivos indicam quais serão as questões que a simulação irá responder, sendo que os planos devem ser traçados de modo a incluir as várias possibilidades que serão analisadas.

Construção do modelo

Uma vez que o sistema a ser simulado está bem definido é necessário abstraí-lo em um modelo, o qual irá representar uma visão particular deste.

Um modelo deve ser de fácil compreensão, sem que haja perda das características importantes e essenciais para determinação do desempenho. A quantidade de detalhes incluídos deve variar de acordo com o propósito de construção do modelo. É recomendado que o modelo seja a princípio bem simples e que os detalhes sejam inseridos até que a complexidade desejada seja alcançada. A seção 2.3 considera as formas de se representar um modelo.

Coleta de dados

Ao gerar a descrição do modelo surge a necessidade de definir quais os tipos dos dados que irão fluir no sistema. Estes serão os dados de entrada e servirão de parâmetros do modelo. Podem ser tanto valores hipotéticos (quando não se tem um sistema de referência) como baseados em alguma análise preliminar (quando o sistema de referência ou algum sistema similar existe). Essa etapa pode ser realizada paralelamente à construção do modelo.

Codificação

A escolha do computador, linguagem e ferramentas dependem dos seguintes fatores: sistema a ser simulado; recursos disponíveis (*hardware* e *software*); preferências do usuário; e descrição do comportamento dinâmico do sistema.

Três abordagens podem ser utilizadas para descrever o comportamento dinâmico de sistemas discretos: atividades, processos e eventos. Um sistema pode ser visto dinamicamente como uma coleção de processos interativos, cada um composto por diversas atividades, com as interações controladas e coordenadas pela ocorrência de eventos. Normalmente as linguagens são orientadas a uma dessas três abordagens.

Para a codificação do modelo, pode-se seguir um dos seguintes enfoques (Santana, 1990; Santana, 1990a):

- Desenvolvimento do programa de simulação em uma linguagem convencional: esta abordagem apresenta como vantagens o fato do usuário não precisar aprender uma nova linguagem e de ser flexível. A desvantagem desta abordagem é a inexistência de ferramentas (estruturas de dados, coleta de estatísticas, emissão de relatórios, etc.) para auxiliar no desenvolvimento da simulação;

- Utilização de um pacote de uso específico para o desenvolvimento do programa de simulação: normalmente são ambientes voltados para a avaliação de sistemas particulares. Possui a vantagem de ser de fácil utilização para o objetivo proposto mas são pouco flexíveis a mudanças;
- Uso de linguagens de simulação de uso geral: são linguagens já projetadas para o desenvolvimento de uma simulação. Possuem a vantagem de livrar o programador de certos detalhes de implementação da simulação mas a desvantagem de se ter a necessidade de aprender uma nova linguagem.
- Uso de uma extensão funcional de uma linguagem de programação de uso geral: são conjuntos de bibliotecas inseridas nas linguagens convencionais compondo um ambiente completo para o desenvolvimento da simulação. Une as vantagens das linguagens convencionais e das linguagens de simulação.

Verificação

A fase de verificação consiste em verificar se o programa de simulação é uma implementação válida do modelo (MacDougall, 1987). Esta fase responde à seguinte pergunta: “O modelo operacional (programa) representa o modelo conceitual (modelo)?”. Para pequenos sistemas, uma inspeção bem realizada no modelo final pode ser suficiente. Para grandes modelos é sugerido que a verificação seja contínua, não esperando que todo o programa esteja pronto para iniciar a verificação. Os meios pelos quais a verificação pode ser realizada são (Banks, 1998): seguir os princípios da programação estruturada (*top-down* ou programação modular); documentar muito bem o programa; mais de uma pessoa analisar o programa; verificar se os valores de entrada estão sendo usados apropriadamente; utilizar um depurador.

Validação

Na fase de validação deve-se demonstrar que o modelo proposto é uma representação do sistema a ser simulado respondendo a seguinte pergunta: “O modelo conceitual pode substituir o sistema real?” (Banks, 1998). A validação pode ser considerada em dois casos: quando o sistema simulado existe e pode ser medido, e quando o sistema modelado não existe, tendo-se apenas o projeto do sistema a ser simulado. No primeiro caso, o objetivo da análise é avaliar uma mudança proposta no sistema e a validação é baseada na comparação dos resultados do modelo com as medidas reais do sistema. No segundo caso, o objetivo da análise é estimar o desempenho do projeto ou mesmo avaliar projetos alternativos. Neste caso, o modelo é validado baseando-se nos resultados esperados no projeto do sistema e cada suposição e abstração são justificadas. Outra possibilidade para avaliação é através de comparações com outros modelos do projeto já validados, como por exemplo, modelos analíticos (MacDougall, 1987), ou através de situações em que os resultados já são conhecidos.

Período experimental

É nesta fase que as decisões com relação ao tamanho da simulação, o número de replicações e a maneira como a simulação é inicializada são tomadas de maneira a se obter as informações desejadas a um custo mínimo. A seção 2.4 considera os métodos que podem ser utilizados para se atingir resultados confiáveis.

Produção das rodadas e análise

As execuções iniciam e os resultados a serem analisados começam a ser coletados. Em simulações estocásticas, como as entradas de uma simulação são baseadas na geração de valores aleatórios e as saídas são funções dessas variáveis de entrada, a análise dos resultados de uma simulação é um problema estatístico. Os métodos para análise destes resultados baseiam-se em estimar o intervalo de confiança dos valores médios obtidos na simulação (MacDougall, 1987). Isto quer dizer que a simulação pára quando a estimativa da variância da média está dentro de uma certa tolerância especificada (intervalo de confiança). Se a precisão não foi atingida é necessária a execução de mais rodadas. Entre os métodos para análise de saída pode-se destacar os métodos regenerativos, por replicação e *batch means*, os quais são descritos na seção 2.4.

Emissão do relatório

Os resultados desejados e obtidos com a simulação devem ser fornecidos ao usuário de uma forma clara e concisa.

Dois passos no desenvolvimento da simulação são detalhados a seguir: a modelagem (seção 2.3) e os métodos para análise dos resultados (seção 2.4). O detalhamento desses dois passos deve-se à importância deles no desenvolvimento deste trabalho. Sendo o objetivo deste trabalho a especificação de um ambiente para desenvolvimento de simulação distribuída, é necessário que o usuário especifique o seu sistema através de um modelo. Na seção 2.3, quatro abordagens são analisadas e uma comparação entre elas é discutida. Para que o usuário não precise se preocupar com a validade dos resultados da simulação, outra fase importante é detalhada na seção 2.4: a análise de saída dos dados obtidos a partir da simulação.

2.3 Modelagem

Para utilizar a simulação é necessário construir e analisar modelos que representem o sistema. O principal passo em um processo de modelagem é abstrair as características do sistema em um modelo (MacDougall, 1987). Um modelador deve ter a capacidade de definir quais características são essenciais e estabelecer uma relação entre essas características. O modelo deve ser uma representação do sistema dentro de uma precisão estipulada através dos objetivos da simulação.

Os modelos podem ser classificados de diferentes formas (Pegden *et al.*, 1995). Uma possível classificação pode ser com relação à quantidade de detalhes inseridos: existem os que são uma *réplica* precisa do sistema a ser simulado (também conhecidos como simuladores, como por exemplo, simuladores de vôo e de carro) e os que podem ter suas *principais características abstraídas*. Nesse caso, a inclusão de muitos detalhes pode introduzir complicações desnecessárias enquanto que a exclusão de características importantes pode invalidar o modelo (Soares, 1992).

Uma outra classificação pode ser considerada quando se verifica a influência ou não de variáveis aleatórias no sistema. Poucos sistemas podem ser considerados totalmente livres da influência da aleatoriedade. Os modelos que representam esses sistemas são conhecidos como *modelos determinísticos*. Os sistemas que consideram a aleatoriedade são classificados como *modelos estocásticos*.

Classificando os modelos com relação ao tempo tem-se os *modelos dinâmicos*, os quais descrevem o comportamento do sistema através do tempo, e os *modelos estáticos*, os quais simplesmente fornecem um retrato do sistema em um ponto específico do tempo. Os modelos dinâmicos podem ainda ser *contínuos* ou *discretos no tempo*. Os modelos de *estados discretos* representam sistemas onde as mudanças nas variáveis de estados ocorrem em pontos específicos e descontínuos do tempo simulado, alterando o estado do sistema espontaneamente. Nos modelos de *estados contínuos* as variáveis de estado variam continuamente ao longo do tempo de simulação e normalmente são descritos através de equações diferenciais.

Este trabalho enfoca os modelos dinâmicos e contínuos no tempo, estocásticos e de estados discretos, uma vez que eles são os que melhor representam os modelos de sistemas computacionais.

Um ponto importante para o desenvolvimento de um modelo é a definição da abordagem a ser utilizada para representar o modelo. As abordagens analisadas neste trabalho para a modelagem de sistemas são: *Statecharts*, Redes de Petri, Teoria de Filas e *Queueing Statecharts*, as quais são descritas a seguir.

2.3.1 Statecharts

Statecharts são uma extensão das máquinas de estado finito (MEF) que permitem a representação de hierarquia, concorrência e comunicação entre os diversos estados de um sistema (Harel, 1987). *Statecharts* têm a finalidade de representar sistemas reativos, ou seja, sistemas que reagem a estímulos internos e externos (Harel *et al.*, 1987).

Os *statecharts* foram propostos com a finalidade de resolver alguns problemas que os diagramas de estado apresentavam em relação à representação de sistemas. Entre os problemas pode-se citar (Harel *et al.*, 1987):

1. Os diagramas de estado não fornecem a noção de profundidade, hierarquia ou modularidade;
2. Conforme o sistema a ser descrito cresce linearmente, o número de estados cresce exponencialmente;
3. Os diagramas de estados não possuem nenhuma característica que permita a representação de estados concorrentes.

Estes problemas são superados através da decomposição dos estados em subestados (decomposições AND/OR) combinado com um eficiente mecanismo de comunicação (Harel *et al.*, 1990).

De uma maneira resumida, tem-se (Harel, 1987):

Statecharts = diagrama de estados + hierarquia + concorrência + mecanismos de comunicação

2.3.1.1 Notação

Statechart utiliza os diagramas de estado para a representação do sistema. Os estados do sistema são denominados **estados** e um determinado evento faz com que seja disparada uma transição, ou seja, ocorre a mudança de um estado para outro.

A sintaxe da transição é:

$$\alpha [C] / \beta$$

onde α é o evento que dispara a transição, C é a condição que garante que a transição não irá ocorrer a não ser que seja verdadeira quando α ocorrer e β é a ação que será executada quando a transição for executada. Eventos e condições podem ser vistos como entradas e ações como saídas (Harel *et al.*, 1987).

2.3.1.2 Hierarquia de estados

A idéia de profundidade, ou hierarquia, pode ser observada na figura 2.2, onde (a) e (b) são equivalentes. Na figura 2.2 (b) os estados A e C foram agrupados em um novo estado D , onde D pode estar em A ou em C (decomposição OR). Pode-se observar que houve um encapsulamento de estados dentro de estados.

Os símbolos e , f , g e h são eventos que disparam as transições e c , entre colchetes, é uma condição. Deste modo $g[c]$ dispara a transição de A para C se g ocorrer e no mesmo momento c for verdadeiro (Harel *et al.*, 1987).

Observa-se na figura 2.2 (b) que a seta f sai do contorno de D e, por definição, pode ser aplicada tanto a A quanto a C . Em (a) observa-se somente uma seta *default* (seta que indica o estado inicial) enquanto que em (b) observa-se duas. Isto se deve ao fato de que a seta *default* que está no estado C indica que C é o estado *default* entre A e C quando já se está em D . Isto

faz com que não haja a necessidade da continuidade da seta h para dentro de D (Harel, 1987; Harel *et al.*, 1987).

Figura 2.2. Decomposição OR

2.3.1.3 Concorrência

Em *statecharts* a concorrência é obtida através da decomposição AND, ou ortogonalidade. Os estados concorrentes são também denominados estados ortogonais pois trata-se de um produto cartesiano de subestados que estão ativos em um determinado momento, como pode ser observado na figura 2.3 (a) (Harel *et al.*, 1990).

Do mesmo modo que na figura 2.2, tem-se na figura 2.3 que (a) e (b) são equivalentes. Ao contrário da decomposição OR, para estar no estado Y deve-se estar em A e D ao mesmo tempo. A entrada ao estado Y implica em uma entrada aos estados *default* $\{B,F\}$. A ocorrência do evento e pode levar ao par $\{C,G\}$ e a ocorrência de k pode levar ao par $\{B,E\}$.

A ortogonalidade é a maneira utilizada pelo *statechart* para solucionar o problema de crescimento exponencial dos estados.

Figura 2.3. Decomposição AND

2.3.2 Redes de Petri

As Redes de Petri foram propostas por Carl Adam Petri em 1.962 como uma ferramenta matemática para modelar sistemas distribuídos e, em particular, notações de concorrência, comunicação e sincronismo (Moore & Brennan, 1995). A partir da década de 70, expandiu-se também para a utilização na modelagem de componentes de *hardware*, linguagens de programação, controle de processos e protocolos de comunicação. Na década de 80 as áreas de sistemas de informação, banco de dados, automação de escritórios e inteligência artificial passaram também a adotá-las (Heuser, 1990). Atualmente, tem-se utilizado Redes de Petri para modelagem de sistemas concorrentes, assíncronos, distribuídos, paralelos, não determinísticos e estocásticos, *software* distribuído, banco de dados distribuídos, controle industrial, memória de multiprocessadores, sistemas *data-flow* e tolerante a falhas, programação lógica e VLSI, compiladores e sistemas operacionais, linguagens formais, redes locais, fatores humanos e redes neurais, entre outros (Brennan *et al.*, 1995).

As Redes de Petri consistem de quatro elementos básicos: *tokens*, lugares, transições e arcos, como pode ser observado na figura 2.4.

Figura 2.4. Elementos básicos de uma Rede de Petri (a) *token*, (b) lugar, (c) transição, (d) arco

Os *tokens* são entidades conceituais, que modelam os objetos que se movem por uma rede real. Lugares representam as localizações onde objetos esperam por processamento. Transições representam processos, eventos ou atividades. Arcos representam os caminhos dos objetos através do sistema (Maciel *et al.*, 1996).

A rede mais simples é um dígrafo direcionado cujos nós podem ser lugares ou transições. Os arcos podem conectar lugares a transições ou transições a lugares e possuem direção, indicando que o fluxo ocorre em uma única direção. Uma Rede de Petri é inicialmente marcada indicando quais *tokens* estão contidos em cada lugar. Uma transição é disparada quando existe pelo menos um *token* em todos os lugares de entrada que estão ligados a essa transição. Se a transição é disparada então um *token* é retirado de cada lugar de entrada conectado a transição e um *token* é adicionado a cada lugar de saída conectado a transição. Uma Rede de Petri é executada quando se estabelecem marcações iniciais e um conjunto de regras de disparo para as transições (Moore & Brennan, 1995). Um exemplo de uma Rede de Petri pode ser observada na figura 2.5

Uma Rede de Petri pode conter também arcos múltiplos ou valorados, ou seja, para que uma transição ocorra é necessário um número n de *tokens* no lugar.

Figura 2.5. Modelo de uma Rede de Petri com os elementos básicos

2.3.2.1 Regras de disparo de Redes de Petri

As regras de disparo das Redes de Petri especificam o comportamento das transições, ou seja, as condições sob as quais processos ou eventos podem ocorrer (Moore & Brennan, 1995a).

Regras de disparo das transições: três regras ditam o disparo das transições:

1. Quando todos os lugares de entrada que estão ligados a transição possuem pelo menos um *token*, a transição está habilitada;
2. Uma vez habilitada, a transição dispara;
3. Quando a transição dispara, exatamente um *token* é removido de cada lugar de entrada e exatamente um *token* é colocado no(s) lugar(es) de saída.

Regras de sincronização: estão associadas às transições e representam o tempo requerido para completar alguma atividade. Podem ser: estocástica; baseada em uma função de probabilidade; um valor computado ou uma constante.

Regras de decisão: estão associadas aos lugares e são utilizadas quando mais que uma transição está habilitada por um mesmo *token* ou conjunto de *tokens*. Podem ser: com prioridade (o *token* partirá pelo caminho com maior probabilidade), com decisão (o *token* partirá pelo caminho baseado na probabilidade associada) e construída (permite ao usuário especificar as condições nas quais o *token* partirá por um caminho particular).

2.3.2.2 Classificação das Redes de Petri

Existem muitas variedades de Redes de Petri, variando do modelo original (redes binárias) até modelos mais complexos (redes coloridas ou temporizadas).

Redes de Petri Binária: são as mais simples. Nesta classe um lugar pode conter no máximo 1 *token* e todos os arcos possuem valor unitário (Maciel *et al.*, 1996).

Redes de Petri Place-Transition: nesta classe um lugar pode conter mais de um *token* e também os arcos podem ser valorados (Maciel *et al.*, 1996).

Redes de Petri com Arco Inibidor: quando uma rede possui transições com arcos inibidores, a regra de disparo é alterada. Neste caso a transição dispara caso não haja nenhum *token* no lugar de entrada (Maciel *et al.*, 1996).

Redes de Petri Hierárquicas: nesta classe não é introduzida nenhuma alteração na estrutura da Rede de Petri, somente na forma de representação gráfica. Estas redes oferecem uma descrição estruturada do sistema, possibilitando sua especificação em níveis de abstração que podem ser refinados quando necessário.

Redes de Petri Coloridas: estas redes foram desenvolvidas para fornecer um método para distinguir os *tokens* nos lugares (Moore & Brennan, 1995). O principal objetivo é a redução do modelo uma vez que vários recursos podem ser representados em uma mesma rede (Maciel *et al.*, 1996).

Redes de Petri Estocásticas: a finalidade desta derivação das redes de petri binárias é proporcionar medidas de desempenho. A função de disparo está associada a um tempo de disparo para cada transição. Os *tokens* não são movidos ao disparo da transição, o que só ocorre depois de decorrido o tempo de disparo associado à transição.

Redes de Petri Temporizada Determinísticas: as redes temporizadas podem ser classificadas em: disparo atômico e disparo de três fases. Nas redes de disparo atômico, assim que uma transição é habilitada um contador é decrementado a uma taxa constante e assim que atinge o valor zero a transição dispara. Nas redes de disparo de três fases, quando uma transição é habilitada, os tokens são retirados do lugar de entrada e são depositados nos lugares de saída somente após o tempo associado à transição disparada.

2.3.3 Redes de Filas

A disputa por um recurso dentro de um sistema é um ponto fundamental na operação do mesmo. Se não há disputa, a análise de desempenho se torna mais fácil. Porém, a ausência de disputa em sistemas reais não é uma suposição realista. Na área de computação tem-se como exemplo processos em um computador competindo pela UCP, unidades de E/S e canais de comunicação; mensagens em redes de comunicação competindo por enlaces, *buffers*, permissões e janelas (Soares, 1992).

A disputa faz com que entidades esperem um certo tempo por serviços. Neste caso, as linhas de espera são denominadas filas e modelagem de filas é uma parte essencial no estudo de simulação.

Os componentes básicos de uma rede de filas são: **servidores**, **filas** e **clientes**. Os recursos modelados no sistema correspondem aos servidores, os quais prestam serviços aos clientes. A fila é uma área de espera para clientes que estão requisitando serviços e não são

atendidos prontamente devido ao(s) servidor(es) estar(em) ocupado(s). O conjunto de um ou mais servidores e uma ou mais filas é denominado **centro de serviço**. O conjunto de centros de serviço é denominado uma **rede de filas**.

A figura 2.6 exemplifica a representação gráfica de um modelo com um centro de serviço de um servidor e uma fila.

Figura 2.6. Centro de Serviço: uma fila e um servidor

Outras variações possíveis de centros de serviço são: centro de serviço com uma fila e vários servidores (figura 2.7a), centro de serviço com várias filas e um servidor (figura 2.7b), centro de serviço com várias filas e vários servidores (figura 2.7c) e servidor infinito (figura 2.7d). O servidor infinito é também conhecido como centro *delay* e possui a característica de não ter fila. Os outros centros de serviço podem ser centros de capacidade fixa ou centros dependentes da carga. No primeiro caso, o tempo de serviço do servidor não depende do número de clientes. Já no segundo caso, o tempo de serviço é dependente do número de clientes (Jain, 1991).

As redes de filas podem ser classificadas como abertas, fechadas ou mistas. Uma rede de filas aberta possui chegadas e partidas externas de clientes, como pode ser observado na figura 2.8a. A rede de filas fechada não possui chegada nem partida e os clientes ficam circulando no sistema (figura 2.8b). A rede de filas mista é aberta para certos clientes e fechada para outros (figura 2.8c) (Jain, 1991).

Figura 2.7. Exemplos de centros de serviço

Figura 2.8. Redes de filas abertas, fechadas e mistas

De forma a analisar os sistemas, algumas características devem ser observadas (Jain, 1991):

Padrão de chegada

O padrão de chegada é a frequência com que os clientes chegam ao sistema. Essa frequência pode ser regular ou aleatória. Se as chegadas forem aleatórias os dados podem ser obtidos através de tabelas ou pela utilização de uma distribuição de probabilidade. Algumas distribuições importantes são (Soares, 1992):

- Distribuição Uniforme;
- Distribuição Triangular;
- Distribuição Exponencial;
- Distribuição de Poisson (mais comum);
- Distribuição Normal;
- Distribuição de Erlang.

Distribuição do serviço e capacidade de serviço

O serviço solicitado por um cliente será denominado **demanda de serviço** ou simplesmente **trabalho**. A unidade de serviço varia dependendo da natureza do servidor e dos

clientes. Pode ser uma instrução se o servidor for uma UCP e o cliente um programa. Se o servidor é uma linha de transmissão, os clientes são mensagens, a unidade de serviço pode ser bit ou byte. A demanda de serviço pelos clientes seguirá uma distribuição que será denominada distribuição de serviço.

Deve-se especificar também a capacidade (ou razão de processamento) do servidor, que significa quão rápido o servidor realiza o serviço. A unidade da capacidade irá também depender do servidor e dos clientes. No caso da UCP, a unidade poderia ser instruções/seg. ou MIPS e no caso da linha de transmissão a unidade poderia ser bps ou kbps.

Se a demanda de serviço de um cliente é S [unidades de serviço] e o correspondente servidor possui capacidade C [unidades de serviço / seg.], então a razão S/C [seg.] é chamada de **tempo de serviço**. Em outras palavras, é o tempo necessário para que um servidor execute um serviço. É também especificado através de uma distribuição de probabilidade.

Disciplina da fila (Disciplina de escalonamento)

A disciplina da fila irá determinar como os clientes são escalonados e servidos. A disciplina mais comum é a *first come, first served* (FCFS) na qual os primeiros clientes a chegar são os primeiros a serem servidos. Existe também a disciplina *last come, first served* (LCFS) que possui a estrutura de pilha e os últimos clientes a chegar são os primeiros a serem atendidos. Na disciplina *Round Robin* (RR), os clientes são atendidos por um pequeno intervalo de tempo, denominado *quantum*. Se esse *quantum* não foi suficiente para o servidor realizar o serviço do cliente, este volta para o fim da fila e aguarda novamente a sua vez.

Algumas vezes, alguns clientes são mais importantes que outros e merecem um tratamento melhor. O procedimento que diferencia alguns clientes dos outros é chamado procedimento de escalonamento com prioridade e o sistema é referido como fila com prioridade. As disciplinas de filas com prioridades podem ser divididas em 2 tipos: preemptiva e não preemptiva. Se um cliente que está sendo servido pode ser interrompido e voltar para a fila sempre que um cliente de maior prioridade entrar na fila então a disciplina da fila é preemptiva. Se o cliente não é interrompido com a chegada de um outro com maior prioridade a disciplina da fila é não preemptiva (Soares, 1992).

A notação padrão para sistemas baseados em redes de filas é: $A/S/c/K/m/SD$, onde A representa a distribuição do tempo de chegada, S representa a distribuição do tempo de serviço, c é o número de servidores, K é o número máximo de clientes no sistema e m é o número de clientes disponíveis na fonte e SD é a disciplina da fila. Os símbolos mais usados para as distribuições são: D (tempo de chegada ou de serviço constante), M (distribuição exponencial), E_k (distribuição k-Erlang), H_k (distribuição hiperexponencial k-estágios) e G (distribuição geral). (MacDougall, 1987; Jain, 1991). Para facilitar a representação, adotou-se que não existe limite para o número de clientes, a fonte é infinita e a disciplina da fila é FCFS.

Desse modo, o modelo mais simples é o M/M/1, com distribuição exponencial de chegada e serviço e um centro de serviço com um único servidor.

2.3.3.1 Redes de Filas Estendidas

A estrutura de Redes de Filas Estendidas do RESQ (*RESearch Queuing Package*) (Sauer *et al.*, 1980) incorpora novos recursos ao modelo básico de Rede de Filas. Esses novos recursos visam melhor representar complexas características dos sistemas reais, que na maioria das vezes não podem ser modeladas com a estrutura básica (Soares, 1992). Além dos centros de serviços (formados por um ou mais servidores e uma ou mais filas), esta estrutura incorpora ainda os elementos gráficos adicionais: servidor sem fila e os centros passivos.

O servidor sem fila é utilizado para representar servidores que não permitem a formação de filas. Neste caso, deve existir uma política de reescalonamento de modo a permitir uma nova tentativa de utilização do recurso. Graficamente, o servidor sem fila é representado por um círculo.

Já os centros passivos representam recursos que não são servidores mas que devem ser alocados aos clientes simultaneamente com outros recursos. Um exemplo de utilização dos centros passivos é o modelo de uma rede Ethernet onde o cabo de comunicação é modelado como um centro passivo. A representação de modelo com um centro passivo denominado canal de comunicação e um outro centro de serviço denominado disco pode ser observada na figura 2.9. Neste caso, o cliente necessita utilizar ao mesmo tempo os recursos disco e canal.

Figura 2.9. Representação de um modelo utilizando centro passivo

2.3.4 Queuing Statecharts

A abordagem Queueing Statecharts é um dos resultados da tese do aluno do grupo de Sistemas Distribuídos e Programação Concorrente (LaSDPC) do ICMC-USP, Carlos Renato Lisboa Francês (Francês, 2001). Objetivando apresentar uma abordagem mais adequada, Francês (2001) propôs uma nova representação a qual une as principais características de Redes de Filas e de *Statechart* Estocásticos (Francês, 1998). A motivação para a elaboração desta nova representação deve-se aos seguintes fatores:

- dificuldade de se representar, em *Statecharts*, o caminho linear que um determinado cliente traça durante a sua passagem pelo sistema, a qual é representada através da ligação sequencial entre os componentes em um modelo de redes de filas.

- dificuldade de se representar adequadamente, com redes de filas, a complexidade que os servidores necessitam para suas descrições, o que pode ser feito de forma natural com *Statecharts*.

Os estados permitidos no Queueing Statecharts são (Francês, 2001):

- Fonte (Source): representa um componente genérico do sistema responsável pela geração de cliente obedecendo a uma distribuição de probabilidade.;
- Fila (Queue): representa a fila de um servidor;
- Servidor (Server): representa o componente responsável pela prestação de um determinado serviço do sistema. Assim como o Source, obedece a uma distribuição de probabilidade;
- Sorvedouro (Destination): representa o componente do sistema que absorve os clientes após a execução do serviço.

Para cada um desses estados é definido um *template*¹ definidos a seguir. As figuras 2.10, 2.11, 2.12, 2.13 e 2.14 apresentam, repectivamente, os *templates* para a fonte, fila, servidor, sorvedouro e centro de serviço.

Figura 2.10 Template para Fonte

Figura 2.11. Template para Fila

¹ No contexto do trabalho de Francês (2001), “*template* é um conjunto de estados (possivelmente unitário) que define o funcionamento básico de um determinado componente do sistema”

Figura 2.12. Template para Servidor

Figura 2.13. Template para Sorvedouro

Figura 2.14. Template para Centro de Serviço

O conjunto de eventos definidos para *Queueing Statecharts* pode ser observado na tabela 2.1 (Francês, 2001).

Tabela 2.1. Eventos-padrão do *Queueing Statecharts* 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_Q _j (<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 _j (<i>Decrease the number of clients</i>)	Decrementa a variável aleatória que representa o

<i>in the Server Queue</i>)	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 próximo cliente pode ir ao servidor.
$eos \{p\}$ (<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.

As figuras 2.15 e 2.16 têm como objetivo ilustrar a diferença gráfica entre os métodos Redes de Filas e *Queuing Statecharts*, apresentando o modelo simplificado de um servidor de arquivos.

Figura 2.15. Servidor de arquivos representado utilizando Redes de Filas

Figura 2.16. Servidor de arquivos representado utilizando *Queueing Statecharts*

Como pode ser observado na figura 2.16, a estrutura do modelo é semelhante à do

modelo de redes de filas, sendo o fluxo dos clientes no sistema facilmente identificado. A vantagem dessa representação é a possibilidade de se expandir a definição da fonte e dos servidores caso seja necessário definir maiores detalhes.

2.3.5 Comparação entre as abordagens apresentadas

Cada uma das abordagens descritas nessa seção (Redes de Filas, Redes de Petri e *Statecharts* e *Queueing Statecharts*) possui características importantes para representação de modelos.

As Redes de Filas são utilizadas para representar sistemas que possuem dois elementos básicos: servidores e clientes. Esta abordagem possui a capacidade de representar as seguintes características: padrão de chegada de um cliente em um servidor, tempo de serviço do servidor e disciplina da fila. No entanto, esta teoria não consegue representar modelos paralelos. Outra desvantagem é que nem todos os elementos de um sistema são cliente ou servidor. A Rede de Filas Estendida acrescenta alguns recursos importantes para a representação dos modelos.

As Redes de Petri e *Statecharts* possuem basicamente a mesma filosofia (conjunto de estados que são ativados por transições) e ambas conseguem representar sistemas com paralelismo. Uma vantagem de *Statecharts* é a capacidade de representar concorrência e paralelismo. A vantagem das Redes de Petri é a possibilidade de saber onde o cliente está no sistema através dos *tokens*. Uma desvantagem da abordagem Redes de Petri é a dificuldade de se representar graficamente os sistemas mais complexos pois há um crescimento exponencial do número de lugares conforme a complexidade do sistema aumenta.

A abordagem *Queueing Statecharts* apresenta grandes vantagens ao unir as principais características das Redes de Filas e *Statecharts*: permite que o usuário saiba o caminho que o cliente irá percorrer no sistema e permite também a especificação mais detalhada de fontes e servidores.

Para a representação do modelo no ambiente definido neste trabalho optou-se pela utilização das abordagens Redes de Filas Estendida e *Queueing Statecharts*. A escolha dessas duas abordagens deu-se pelo fato de que grande parte dos sistemas de computação que podem ser simulados apresentam a característica de possuir servidores e filas. Além disso, a abordagem *Queueing Statecharts*, por apresentar uma semântica parecida com a dos *Statecharts*, consegue representar o conceito de paralelismo, muitas vezes necessário nos modelos simulados.

2.4 Análise de Saída

As medidas de desempenho ajudam o usuário a entender o comportamento do sistema

sob diversas condições. Normalmente, em sistemas que possuem clientes, servidores e filas, ou seja, em sistemas baseados na teoria de redes de filas, as medidas de desempenho mais utilizadas são (Soares, 1992):

- Utilização do servidor: fração de tempo que o servidor está ativo;
- *Throughput* ou vazão: número de serviços completos por unidade de tempo;
- Tamanho médio da fila: número médio de usuários que esperaram para serem atendidos pelo servidor;
- Tempo médio de espera na fila: razão entre o tempo total de espera na fila e o número de clientes na fila;
- Tempo médio de serviço: razão entre o tempo que o servidor foi utilizado e o número total de clientes servidor;
- Tempo médio no sistema: tempo médio que os clientes gastaram no sistema, contando o tempo de espera na fila e o tempo de serviço;

Como foi observado na seção 2.3, os modelos podem ser classificados em determinísticos ou estocásticos, de acordo com a existência ou não de aleatoriedade nos parâmetros de entrada. Quando a simulação estocástica é considerada, para que as medidas de desempenho tenham significado é necessário que sejam realizados testes estatísticos de modo a garantir que a aleatoriedade dos dados de entrada não interfira nos dados de saída. Assim, a análise de saída, ou seja, a análise dos resultados produzidos pela simulação, é de fundamental importância, uma vez que estes são funções de variáveis aleatórias. Caso a aleatoriedade dos resultados seja ignorada, estes podem não representar o comportamento real do sistema. Kelton (1997) classifica esses sistemas como RIRO (*Randon In, Random Out*) em contraposição aos modelos DIDO (*Deterministic In, Deterministic Out*).

Alguns conceitos e técnicas para estimar e controlar a precisão dos resultados da simulação são considerados nesta seção.

2.4.1 Tipos de Simulação

Dois tipos de simulação devem ser considerados quando se deseja realizar a análise de saída (MacDougall, 1987; Pawlikowski, 1990; Sanchez, 1999):

- **Simulação terminante (horizonte finito):** a ocorrência de algum evento especial determinado pelo próprio sistema é considerado como a indicação de que a simulação deve ser interrompida. É utilizada quando se deseja observar o comportamento de um sistema por um determinado período.
- **Simulação não terminante ou Simulação *Steady-State* (horizonte infinito):** não existe nenhum evento responsável pela interrupção da simulação. Teoricamente,

ela pode ser executada infinitamente até que a precisão desejada seja atingida. É utilizada quando se deseja avaliar o comportamento de sistemas após um longo período.

Devido ao objetivo de avaliar o desempenho dos sistemas computacionais, a maioria das simulações desses sistemas são simulações não terminantes.

2.4.2 Geração de números aleatórios

Um tópico importante que deve ser levado em consideração quando se realiza uma simulação é a geração de números aleatórios. Isto ocorre porque a simulação utiliza várias funções de distribuição de probabilidade, tais como exponencial e normal, e essas funções utilizam as rotinas de geração de números aleatórios. Desse modo, pode-se dizer que o processo divide-se em duas fases: a geração de números aleatórios uniformemente distribuídos entre 0 e 1 e a transformação destes números de acordo a distribuição de probabilidade desejada (Jain, 1991).

Atualmente, a maneira mais conveniente e segura de gerar números aleatórios é através de algoritmos determinísticos, ou seja, através de algoritmos que produzem uma seqüência de números que não é aleatória de fato mas que se comporta como aleatória. Essas seqüências são chamadas de pseudo-aleatórias e os programas que geram essas seqüências são chamados de geradores de números pseudo-aleatórios. No entanto, no contexto da simulação, os números pseudo-aleatórios são denominados de aleatórios por uma questão de simplificação da linguagem (L'Ecuyer, 1998).

Os geradores mais utilizados e conhecidos são os LCG (*Linear Congruential Generator*) onde um número da seqüência é uma função do número anterior, seguindo a fórmula:

$$x_n = (ax_{n-1} + b) \bmod m$$

onde $m > 0$, $a > 0$ e b são números inteiros positivos denominados *módulo*, *constante multiplicativa* e *incremento*. O número x_0 que inicia a seqüência é denominado *semente*. A seqüência possui tamanho igual a i quanto $x_i = x_0$, para $i > 0$. O valor i é denominado *período*. Devido ao fato que os valores x_i são restos de uma divisão por m , existem somente m possíveis valores.

A escolha dos valores das constantes a , c e m afeta diretamente os valores do período e da autocorrelação. Vários pesquisadores estudaram essa influência e chegaram à conclusão que para obter um máximo período deve-se satisfazer as seguintes condições: $m = 2^k$, $a = 4c + 1$ e b ser ímpar (Jain, 1991).

2.4.3 Medidas utilizadas

A maioria das vezes o objetivo da simulação é estimar a média verdadeira μ_x de um dos parâmetro descritos no início desta seção a partir de uma seqüência de observações $x_1, x_2, \dots, x_n$, através do cálculo da média:

$$\bar{X}(n) = \sum_{i=1}^n \frac{x_i}{n} \quad 2.1$$

Cada execução da simulação com uma seqüência de entrada diferente (ou seja, com diferentes sementes) produzirá valores diferentes para $\bar{X}(n)$. Se n tende ao infinito, $\bar{X}$ tende para um valor limitante $E(\bar{X}(n))$ denominado esperança de X e igual à média verdadeira μ_x . Pode-se dizer que neste caso, $\bar{X}$ é um estimador não viciado de μ_x .

Outro estimador não viciado bastante utilizado é o da variância de $\bar{X}(n)$, definido por:

$$\hat{\sigma}^2[\bar{X}(n)] = \sum_{i=1}^n \frac{\{x_i - \bar{X}(n)\}^2}{n(n-1)} = \frac{S_n^2}{n} \quad 2.2$$

onde S_n^2 é a variância amostral.

Outras medidas podem ser necessárias, como por exemplo, as que medem a qualidade dos valores estimados. Duas medidas são utilizadas para medir a precisão do valor estimado (Pawlikowski, 1990):

- *Bias*: mede o desvio do valor estimado em relação ao valor real do parâmetro:

$$Bias[\bar{X}(n)] = E[\bar{X}(n) - \mu_x] \quad 2.3$$

- *Mean Square Error (MSE)*: erro médio quadrado do valor estimado:

$$MSE[\bar{X}(n)] = E\{[\bar{X}(n) - \mu_x]^2\} = \{Bias[\bar{X}(n)]\}^2 + \{\sigma^2[\bar{X}(n)]\}^2 \quad 2.4$$

A precisão com que o valor $\bar{X}(n)$ se aproxima da média verdadeira μ_x pode ser calculada através da probabilidade:

$$P(|\bar{X}(n) - \mu_x| \leq \Delta_x) = 1 - \alpha \quad 2.5$$

ou

$$P(\bar{X}(n) - \Delta_x \leq \mu_x \leq \bar{X}(n) + \Delta_x) = 1 - \alpha \quad 2.6$$

onde

$$\Delta_x = t_{n-1, 1-\alpha/2} \hat{\sigma}[\bar{X}(n)] \quad 2.7$$

é metade do intervalo de confiança do parâmetro estimado, $1 - \alpha$ é o nível de confiança ($0 < \alpha < 1$) e $t_{n-1, 1-\alpha/2}$ é o ponto crítico no valor $1-\alpha/2$ da distribuição *t-Student* com $(n-1)$ graus de liberdade (MacDougall, 1987; Pawlikowski, 1990).

2.4.4 Período Transiente ou Período de *Warm-Up*

O período transiente equivale ao período em que o sistema ainda não está estatisticamente em equilíbrio. Isto implica que, se o sistema for estável, após este período os resultados da simulação tenderão a um período de equilíbrio. Caso o sistema seja instável, a simulação nunca ultrapassará o período transiente e os resultados permanecerão numa trajetória não estacionária, alterando seu comportamento continuamente.

Já que as observações coletadas neste período não representam um comportamento real do sistema simulado, seria natural que elas fossem eliminadas e não consideradas nos resultados finais. No entanto, caso essas observações sejam descartadas pode-se aumentar a variância (fórmula 2.2) e conseqüentemente aumentar o valor do erro médio quadrado (fórmula 2.4). Por outro lado, ignorar o período transiente implica no aumento da *bias* (fórmula 2.3), pois os valores são fora da realidade (Pawlikowski, 1990). A escolha entre a permanência ou eliminação das observações coletadas no período transiente depende das exigências de precisão e da limitação dos recursos disponíveis para a execução da simulação, como por exemplo, o número máximo de observações que podem ser armazenadas.

Caso a opção seja pela eliminação do período transiente um outro problema surge: o tamanho do período transiente. Algumas regras foram estabelecidas de modo a determinar quando o sistema entra em equilíbrio. Maiores detalhes sobre essas regras podem ser obtidos em Pawlikowski (1990). No capítulo 5 será explicada a regra que foi adotada para o desenvolvimento do protótipo deste trabalho.

2.4.5 Correlação entre as observações

Estimar um parâmetro através de um intervalo de confiança é a principal função da fase de análise de saídas. No entanto, o maior problema encontrado na análise dos resultados é o alto grau de correlação entre as observações, ou seja, a condição de independência nem sempre é válida. Se as observações são correlacionadas as fórmulas apresentadas até aqui não podem ser utilizadas.

Isto se deve ao fato de que, quando são utilizados valores independentes, como por exemplo, dois valores x_i e x_{i+1} , estes fornecem duas diferentes porções para a construção da estimativa. No entanto, se os valores x_i e x_{i+1} forem correlacionados eles acrescentam muito pouco na construção da estimativa.

A correlação entre k_b lotes de tamanho m pode ser medida pelo estimador do coeficiente de autocorrelação, definido por:

$$\hat{r}(k, m) = \frac{\hat{R}(k, m)}{\hat{R}(0, m)} \quad 2.8$$

onde,

$$\hat{R}(k, m) = \frac{1}{k_b - k} \sum_{i=k+1}^{k_b} [\bar{X}_i(m) - \bar{X}(n)][\bar{X}_{i-k}(m) - \bar{X}(n)] \quad 2.9$$

é o estimador da autocovariância de ordem $k=0, 1, 2, \dots$ da seqüência de médias de lotes. É importante notar que este estimador é definido como a esperança do produto dos desvios das médias dos valores espaçados de k observações e que

$$\hat{r}(0, m) = E[(\bar{X}_i(m) - \bar{X}(n))(\bar{X}_i(m) - \bar{X}(n))] = E[(\bar{X}_i(m) - \bar{X}(n))^2] = \sigma^2(\bar{X}(n))$$

A seqüência $\{\hat{r}(k, m)\}$ é definida como seqüência de correlação, sendo que:

$$-1 \leq \hat{r}(k, m) \leq 1$$

Quanto mais próximo $\hat{r}(k, m)$ estiver de 1 ou -1 , mais correlacionadas serão as observações. Se $\hat{r}(k, m) = 0$ pode-se dizer que as observações não são correlacionadas.

A correlação entre as observações pode ser melhor estimada através dos estimadores de jackknife (Pawlikowski, 1990 *apud* Miller, 1974). Um estimador de jackknife do coeficiente de correlação de ordem k para um seqüência de lotes de tamanho m é calculado da seguinte maneira:

$$\hat{\hat{r}}(k, m) = 2\hat{r}(k, m) - \frac{\hat{r}'(k, m) + \hat{r}''(k, m)}{2} \quad 2.10$$

onde os estimadores do lado direito da equação são calculados como estimadores descritos nas fórmulas 2.8 e 2.9, exceto que $\hat{r}(k, m)$ é calculado sobre todas as k_b médias dos lotes e $\hat{r}'(k, m)$ e $\hat{r}''(k, m)$ são calculados, respectivamente, sobre a primeira e a segunda metade das médias.

2.4.6 Métodos para Análise de Saída

Vários métodos têm sido propostos com a finalidade de tratar os problemas que surgem com o período de *warm-up* e com a correlação existente nas observações coletadas durante o período de equilíbrio. Esses métodos tentam: (a) enfraquecer ou mesmo remover a correlação ou (b) levam em consideração a correlação e tentam estimá-la. Os métodos mais conhecidos são (MacDougall, 1987; Pawlikowski, 1990):

- Replicação/Deleção;

- Média dos Lotes (*Batch Means*);
- Média dos Lotes sobrepostos (*Overlapping Batch Means*);
- Regenerativo
- Padronização de séries temporais
- Autoregressão;
- Análise Espectral

sendo que os cinco primeiros tem como objetivo enfraquecer ou remover a correlação e os dois últimos tentam estimá-la.

Os métodos replicação/deleção, *batch means*, padronização de séries temporais e análise espectral serão descritos a seguir por serem os utilizados neste trabalho.

2.4.6.1 Método da Replicação/Deleção

No caso das simulações terminantes, o problema do período transiente pode ser solucionado seguindo uma das seguintes abordagens (MacDougall, 1987):

- prevenindo-o, estabelecendo as condições iniciais que representem o comportamento do sistema no estado de equilíbrio;
- eliminando os efeitos apagando as primeiras l observações iniciais, ou
- executando a simulação por um tempo razoavelmente longo de forma a reduzir os efeitos a um nível insignificante.

A primeira opção torna-se praticamente inviável primeiramente por não ser possível prever os valores do estado de equilíbrio e em segundo lugar, caso isso seja possível, por exigir um razoável esforço de programação para estabelecer essas condições. Nesta abordagem de replicação, utiliza-se um método para identificar o tamanho do período transiente e eliminá-lo.

O problema da correlação entre as observações pode ser solucionado realizando-se independentes replicações da simulação. A simulação é repetida um número de vezes e cada replicação utiliza uma sequência diferente e independente de números aleatórios. Supondo que sejam executadas k replicações independentes e que cada replicação produza n observações $(X_{i1}, X_{i2}, \dots, X_{in})$, e sejam descartadas as primeiras l observações, então as médias amostrais

$$Y_i = \frac{1}{n-l} \sum_{j=l+1}^n X_{ij} \quad 2.11$$

são variáveis aleatórias independentes e identicamente distribuídas (*iid*) e

$$\bar{Y}_k = \frac{1}{k} \sum_{i=1}^k Y_i \quad 2.12$$

é um estimador não viciado da média μ e a variância amostral dos $\bar{Y}_i$'s

$$S_k^2(Y) = \frac{1}{k-1} \sum_{i=1}^k (Y_i - \bar{Y}_k)^2 \quad 2.13$$

é um estimador não viciado da variância de $\bar{X}(n)$.

Para k e n suficientemente grandes e um nível de confiança de $(1 - \alpha)$, a média μ estará contida no intervalo de confiança:

$$\bar{Y}_k \pm t_{k-1, 1-\alpha/2} \sqrt{S_k^2/k} \quad 2.14$$

Este método é mais sensível em relação às observações coletadas no período transiente do que os métodos que são baseados em uma única execução pois a cada replicação que se inicia, um novo período transiente acontece (Pawlikowski, 1990).

Algumas observações importantes sobre os valores de l , n e k são (Alexopoulos & Seila, 1998):

1. Se l aumenta para um valor fixo de n , o erro sistemático em cada Y_i devido às condições iniciais diminui mas o erro amostral aumenta devido ao número pequeno de observações (a variância de Y_i é proporcional a $1/(n-1)$);
2. Se n aumenta para um valor fixo de l , os erros sistemático e amostral em Y_i diminuem;
3. O erro sistemático na média amostral Y_i não é reduzido aumentando o número de replicações k .

2.4.6.2 Método da Média dos Lotes (BM – Batch Means)

No caso de simulações não terminantes o problema da correlação é mais difícil de ser tratado. Após o período transiente a simulação entra no período *steady-state* e é deste período que as observações devem ser consideradas na análise final caso tenha-se optado pela eliminação das observações coletadas no período transiente.

Ao contrário do método das replicações, este método é baseado em uma única e longa execução da simulação para a estimação do intervalo de confiança. A sequência original de n observações $(X_1, X_2, \dots, X_n)$ é dividida em k lotes não sobrepostos de tamanho m . As médias desses lotes (*batch means*) podem então ser utilizadas para o cálculo do intervalo de

confiança. Considerando-se k lotes de m observações cada um, então o i -ésimo lote será composto pelas seguintes observações:

$$X_{(i-1)m+1}, X_{(i-1)m+2}, \dots, X_{im} \quad 2.15$$

para $i = 1, 2, \dots, k$ e a i -ésima média é dada por:

$$Y_i(m) = \frac{1}{m} \sum_{j=1}^m X_{(i-1)m+j} \quad 2.16$$

A grande média dos lotes é definida por:

$$Y_k = \bar{X}_n = \frac{1}{k} \sum_{i=1}^k Y_i(b) \quad 2.17$$

a variância por:

$$S_B^2(n, k) = \frac{1}{k-1} \sum_{i=1}^k (Y_i(b) - \bar{Y}_k)^2 \quad 2.18$$

e o intervalo de confiança por:

$$\bar{Y}_k \pm t_{k-1, 1-\alpha/2} \sqrt{S_B^2(n, k)/k} \quad 2.19$$

Se após essas n observações o intervalo obtido alcançou a precisão desejada então a simulação é encerrada, caso contrário um novo lote é obtido e a média final é recalculada (MacDougall, 1987).

Este método se baseia na suposição de que as observações separadas por um espaço de tempo são menos correlacionadas (Pawlikowski, 1990). No entanto, selecionar o tamanho ideal do lote de modo a garantir a não correlação entre os resultados é um grande problema. Uma solução é estimar a correlação entre as médias dos lotes considerando inicialmente o tamanho do lote como sendo m . Se a correlação não puder ser ignorada então se incrementa o tamanho do lote e repete-se o teste. Conclui-se que para utilizar este método são necessárias duas fases: uma para determinar o tamanho do lote e outra para determinar a precisão das estimativas (Pawlikowski, 1990). A autocorrelação entre as médias dos lotes pode ser medida utilizando-se as fórmulas 2.8 ou 2.10.

2.4.6.3 Método da Padronização de Séries Temporais (STS – Standardized Time Series)

Este método foi proposto inicialmente por Schruben (1993) e assume-se que o processo $\{X_i\}$ é *phi-mixing*. Informalmente, um processo $\{X_i\}$ é *phi-mixing* se X_i e X_{i+j} são aproximadamente independentes para valores grandes de j (Alexopoulos, 1994).

Suponha que a seqüência de observações $X_1, \dots, X_n$ seja dividida em k lotes, cada um com tamanho b e que $Y_1(b), \dots, Y_k(b)$ sejam as médias desses lotes.

Para um valor suficientemente grande de n , as variáveis aleatórias:

$$A_i = \sum_{j=1}^b \left(\frac{n+1}{2} - j \right) X_{(i-1)b+j} \quad i = 1, \dots, k \quad 2.20$$

tornam-se variáveis independentes e igualmente distribuídas (iid), segundo a distribuição normal.

De acordo com este método, a variância σ^2 pode ser estimada a partir da seguinte fórmula:

$$\hat{\sigma}(A^2) = \frac{12}{(b^3 - b)k} \sum_{i=1}^k A_i^2 \quad 2.21$$

e o intervalo de confiança por:

$$\bar{Y}_k \pm t_{k,1-\alpha/2} \sqrt{\hat{\sigma} / n} \quad 2.22$$

2.4.6.4 Análise Espectral (SA – Spectral Analysis)

O método análise espectral foi proposto por Fishman (1967). Este método assume que as observações representam a seqüência correlacionada $X_1, X_2, \dots, X_n$, e transforma a análise para o domínio da freqüência, através de uma transformação de Fourier na função de autocorrelação $\{R(k)\}$, $k = 0, 1, 2, \dots$, chegando na função de densidade espectral:

$$p_x(f) = R(0) + 2 \sum_{j=1}^{\infty} R(j) \cos(2\pi f j) \quad 2.23$$

para $-\infty \leq f \leq +\infty$. Devido à aleatoriedade das observações coletadas, a função de análise espectral é também uma função aleatória.

Para n suficientemente grande:

$$\sigma^2[\bar{X}(n)] \cong \frac{p_x(0)}{n} \quad 2.24$$

ou seja, o estimador da variância amostral pode ser obtido a partir do estimador de $p_x(f)$ quando $f = 0$.

Após definido como a variância pode ser obtida, o problema passa a ser achar um bom estimador da função de densidade espectral. Várias pesquisas são voltadas a esse objetivo (Pawlikowski, 1990). Uma das alternativas é estimar $p_x(0)$ a partir do periodograma $\{\bar{X}(j/n)\}$, $j = 0, 1, \dots$, da seqüência de observações $x_1, x_2, \dots, x_n$. Esse periodograma é uma

função da transformada discreta de Fourier $\{A_x(j)\}$ das observações $x_1, x_2, \dots, x_n$. Desta forma:

$$\Pi\left(\frac{j}{n}\right) = \frac{|A_x(j)|^2}{n} \quad 2.25$$

e

$$A_x(j) = \sum_{s=1}^n x_s \exp\left[-\frac{2\pi i(s-1)j}{n}\right] \quad 2.26$$

onde $i = \sqrt{-1}$.

Para encontrar um estimador não viciado de $p_x(0)$, o periodograma é transformado em uma função *smoother* no logaritmo da média do periodograma:

$$L(f_i) = \log\left\{\left[\Pi\left(\frac{2j-1}{n}\right) + \Pi\left(\frac{2j}{n}\right)\right]/2\right\} \quad 2.27$$

Da mesma forma que nos outros métodos, o intervalo de confiança pode ser obtido a partir de:

$$\bar{X}(n) \pm t_{k,1-\alpha/2} \hat{\sigma}_{sp}[\bar{X}(n)] \quad 2.28$$

assumindo que:

$$\hat{\sigma}_{sp}[\bar{X}(n)] = \frac{p_x(0)}{n} \quad 2.29$$

2.4.7 Comparação entre os métodos descritos

A justificativa para a escolha da utilização dos métodos descritos acima se deve aos fatos descritos a seguir. Os métodos de replicação/deleção e média dos lotes são os mais fáceis de serem implementados, oferecem bons resultados na maioria dos casos (quando as observações não são muito correlacionadas) e são conseqüentemente os mais utilizados. O método da padronização de séries temporais, apesar de mais complexo, tanto conceitualmente quanto em termos de implementação, é uma opção atrativa para os casos em que se tem dados com um certo grau de correlação.

Os métodos média dos lotes e padronização de séries temporais utilizam o conceito de lotes para a coleta das observações. Essa característica implica na existência de uma fase, denominada fase de determinação do tamanho do lote, que se inicia logo após encerrado o período transiente e determina o tamanho do lote de modo a garantir que as médias das

observações coletadas sejam suficientemente não correlacionadas. Trata-se de uma fase adicional na simulação e, em alguns casos, implica numa considerável sobrecarga de tempo.

Em alguns casos, que serão tratados neste trabalho, é importante a utilização de um método não baseado em lotes. A análise espectral utiliza as transformadas de *Fourier* e em casos onde a determinação do tamanho do lote despende muito tempo este método se torna atrativo.

2.5 Considerações Finais

A avaliação de desempenho de sistemas computacionais pode ser medida através de diversas técnicas, sendo que cada uma possui vantagens e desvantagens. A simulação é uma técnica amplamente utilizada devido principalmente a seu baixo custo e flexibilidade, uma vez que as mudanças no sistema são facilmente refletidas no modelo.

Existem algumas fases para o desenvolvimento de uma simulação que devem ser seguidas. Dentre essas, a modelagem (abstração do sistema em um modelo) e a análise de saída (avaliação dos resultados da simulação) são duas fases importantes, as quais devem ser seguidas tanto na simulação sequencial como na simulação distribuída, foco deste trabalho.

Dentre as abordagens para a modelagem de sistemas descritas neste capítulo, as abordagens Redes de Filas e *Queueing Statechart* são as utilizadas no desenvolvimento deste trabalho. Destaca-se a abordagem *Queueing Statechart* desenvolvida em um projeto de doutorado de um aluno do Grupo de Sistemas Distribuídos e Programação Concorrente do ICMC-USP e que tem por objetivo unir as características de redes de filas e de *statecharts*.

Os métodos para análise de saída descritos neste capítulo (replicação, *batch means*, padronização de séries temporais e análise espectral) são também os utilizados no desenvolvimento do trabalho.

Definido o conceito de simulação e suas principais fases, o próximo capítulo introduzirá o conceito de simulação distribuída/paralela.

Simulação Discreta Distribuída / Paralela

3.1 Considerações Iniciais

Segundo Fujimoto (2000), duas classes de aplicações de simulação vêm atualmente recebendo maior atenção: as simulações analíticas e os ambientes virtuais.

A simulação analítica é a visão clássica de simulação, ou seja, é a simulação descrita no capítulo anterior. O objetivo desse tipo de simulação é a análise de desempenho dos sistemas complexos, possuindo pouca interferência do usuário durante a execução da simulação, sendo este um observador externo. O tempo da simulação pode avançar mais rápido ou mais lentamente do que o tempo de execução, porém esse tempo de execução deve ser “o menor possível”.

Os ambientes virtuais são caracterizados pela total interferência do usuário no decorrer da execução da simulação. Essa classe possui como objetivo criar uma representação realista do ambiente a ser simulado. Eles diferem das simulações analíticas nos seguintes pontos: primeiramente no que se refere à interação com o usuário, onde é criada uma impressão de *look and feel*, fazendo o usuário se sentir inserido no sistema e participante das decisões; em segundo lugar, o tempo da simulação deve acompanhar a mesma taxa de avanço de tempo percebida pelo usuário. A figura 3.1 ilustra as diferenças entre essas duas classes.

Figura 3.1. Relação entre as classes de aplicação de simulação

A adoção da computação paralela/distribuída nessas duas classes visa a objetivos distintos. Para o caso da simulação analítica, a simulação sequencial apresenta uma desvantagem quando se deseja simular modelos mais complexos, com maiores detalhes: o

tempo de simulação. Devido à aleatoriedade das variáveis de entrada, o programa de simulação deve ser executado diversas vezes de modo a garantir que esta aleatoriedade não interfira nos resultados da simulação. Neste caso, o principal objetivo é reduzir o tempo de execução da simulação e duas soluções vêm sendo estudadas: SRIP (*Single Replication in Parallel*) (Misra, 1986; Fujimoto, 1990) e MRIP (*Multiple Replication in Parallel*) (Heidelberger, 1988; Ewing *et al.*, 1999). Essas duas técnicas são descritas nas seções 3.2 e 3.3. Esta área é também denominada PADS (*Parallel and Distributed Simulation*) em referência ao *Workshop on Parallel and Distributed Simulation*, onde são publicados os principais resultados de pesquisas na área, ou por PDES (*Parallel Discrete Event Simulation*).

Os ambientes virtuais utilizam a computação distribuída com dois objetivos: unir computadores geograficamente distribuídos e integrar simuladores que executem em computadores de diferentes fabricantes, permitindo assim que muitas pessoas, localizadas em lugares distintos, formem um ambiente sintético comum. Os principais usuários dos ambientes virtuais são os departamentos militares, que utilizam essa técnica para desenvolver meios mais efetivos e econômicos de treinamento de recursos humanos, além de ser mais seguro e ecologicamente correto.

O projeto SIMNET (*SIMulator NETworking*) foi um dos primeiros projetos de simulação distribuída em ambientes virtuais e foi desenvolvido pelo DARPA (*Defense Advanced Research Projects Agency*) nos Estados Unidos entre 1983 e 1989. Este projeto viabilizava a interconexão de simuladores autônomos (como por exemplo, simuladores de tanques de guerra) em exercícios de treinamento. O protocolo DIS (*Distributed Interactive Simulation*) foi desenvolvido a partir do SIMNET e definia padrões para suportar interoperabilidade entre simuladores autônomos de treinamento em ambientes de simulação geograficamente distribuídos (Page, 1999).

Uma segunda evolução do SIMNET foi o ALSP (*Aggregate Level Simulation Protocol*) que aplica os conceitos de interoperabilidade do SIMNET a simulações de jogos de guerra. Um importante passo nesta evolução foi a especificação HLA (*High Level Architecture*). Esta arquitetura foi proposta e desenvolvida para suportar reuso e interoperabilidade das simulações do Departamento de Defesa dos EUA (U.S. DoD) e representa a integração dos protocolos DIS e ALSP em uma única arquitetura (Page, 1999). A especificação HLA define serviços e padrões de interface que devem ser usados pelos componentes que pertencem ao modelo.

As pesquisas nas duas classes classificadas por Fujimoto (2000) seguiram caminhos independentes devido principalmente às perspectivas e objetivos distintos. Isto aconteceu até a definição da arquitetura HLA. Atualmente, essas áreas estão se expandindo de modo a se sobreporem em alguns pontos, como por exemplo, na utilização por parte dos ambientes virtuais de algoritmos de sincronização e gerenciamento do tempo, resultados de anos de pesquisa em PADS (Wilson & Weatherly, 1994) e na adaptação de ambientes/linguagens de

simulação desenvolvidas no PADS, para o padrão HLA (Wildt & Wagner, 2001).

A simulação distribuída utilizada nesta tese é a classificada por Fujimoto (2000) como simulação analítica, sendo esta referenciada a partir deste ponto como simulação distribuída. Alguns conceitos básicos sobre computação paralela/distribuída necessários para o entendimento dos conceitos envolvidos em simulação distribuída são apresentados no Apêndice A, sendo que neste capítulo são apresentadas as abordagens SRIP e MRIP e algumas alterações no diagrama de desenvolvimento de uma simulação para o emprego destas abordagens.

3.2 *Single Replication in Parallel - SRIP*

Realizar uma simulação discreta distribuída não é uma tarefa fácil. Isto pode ser observado analisando as estruturas básicas de uma simulação seqüencial:

1. As variáveis de estado que armazenam o estado do sistema;
2. Uma lista de eventos que armazena todos os eventos que foram escalonados mas ainda não simulados;
3. Um relógio global que controla o andamento da simulação.

Cada evento contém um *timestamp* que denota quando uma mudança no sistema irá ocorrer. A lista de eventos é classificada em ordem crescente do tempo de chegada. O programa de simulação consiste basicamente de um *loop* principal que repetidamente retira o evento com menor *timestamp* da lista de eventos e processa este evento. Processar um evento implica em executar algum código para efetivar a mudança de estado e, se necessário, escalonar novos eventos para o futuro. É muito importante notar nesta abordagem que o evento com menor *timestamp* (E_{min}) deve ser o próximo evento a ser processado. Isto garante que os eventos sejam simulados em ordem cronológica no tempo de simulação (MacDougall, 1987). Se um evento (E_x) que contém um *timestamp* maior que E_{min} for processado, este pode alterar as variáveis utilizadas por E_{min} permitindo que o futuro influencie o passado. Erros desse tipo são chamados de *erros de causa e efeito* (Fujimoto, 1990).

A natureza seqüencial da lista de eventos e o relógio global limitam o potencial de paralelismo da simulação. A solução é, portanto, eliminar a lista de eventos em sua forma tradicional, e este é o objetivo da simulação *SRIP* (Reed *et al.*, 1988).

Na simulação *SRIP* o sistema é visto como um conjunto de processos lógicos $PL_0, PL_1, \dots, PL_n$ cada um representando um processo físico. Toda a interação entre os processos físicos é modelada por mensagens (eventos com *timestamp*) enviadas entre os correspondentes processos lógicos. Cada processo lógico contém uma parte do estado correspondente ao processo físico a que corresponde, bem como um relógio local que denota quanto a simulação avançou.

Em uma simulação *SRIP*, garante-se que erros de causalidade não ocorrem se e somente se cada processo lógico executa eventos em ordem não decrescente de *timestamp*.

Considere o seguinte exemplo: dois eventos, E_1 no processo lógico PL_1 e *timestamp* 10 e E_2 no processo lógico PL_2 e *timestamp* 20 (figura 3.2.a). Se E_1 escalona um novo evento E_3 para o PL_2 com *timestamp* menor do que 20, E_3 pode afetar E_2 , sendo necessária a execução sequencial dos três eventos (figura 3.2.b). Se um PL não tem informações sobre qual evento pode ser escalonado por outro evento, conclui-se que o único evento que pode ser escalonado é o de menor *timestamp*, gerando uma simulação sequencial.

Figura 3.2. Ocorrência de erros de causa e efeito

3.2.1 Protocolos para SRIP

Para evitar que os erros de causa e efeito ocorram, diversos mecanismos têm sido propostos. Os mecanismos de sincronização podem ser divididos basicamente em duas categorias: *conservativos* e *otimistas*. Os protocolos conservativos impedem a possibilidade de qualquer erro de causalidade determinando quando é seguro processar um evento, ou seja, determinando quando todos os eventos que poderiam afetar os outros já foram processados. Já os protocolos otimistas utilizam a técnica de *deteção e recuperação*. Neste caso, os erros de causalidade são detectados e um mecanismo denominado *rollback* é utilizado na recuperação desse erro (Fujimoto, 1990).

Alguns autores citam três outras classes: os adaptativos, os mistos e os sem sincronização. Os adaptativos podem mudar dinamicamente seu comportamento no decorrer da simulação, de acordo com a evolução da simulação (Tay *et al.*, 1997). No caso dos protocolos mistos podem existir submodelos executando protocolos otimistas ou conservativos (Bagrodia *et al.*, 1998). Nos protocolos sem sincronização os erros de causalidade são ignorados (Thondugulam *et al.*, 1999).

Os protocolos mais estudados, implementados e utilizados são os conservativos e os otimistas, e devido a estes fatos, são descritos nas seções 3.2.2 (conservativo) e 3.2.3 (otimista).

3.2.2 Protocolos Conservativos

Historicamente, o primeiro mecanismo de sincronização para simulação distribuída seguia a abordagem conservativa e foi desenvolvido independentemente por Chandy e Misra (1979) e por Bryant (1977), sendo então denominado CMB (Chandy, Misra e Bryant).

A principal característica deste protocolo é a determinação de quando é seguro executar um evento, ou seja, o evento só é executado quando há garantias de que nenhum outro evento com *timestamp* menor chegará para ser executado. Enquanto isso não é garantido, o processo fica bloqueado o que pode gerar uma queda no desempenho e a situações de *deadlock*.

Os mecanismos conservativos diferem entre si na maneira como o *deadlock* é tratado. Alguns previnem o *deadlock* e outros deixam que ele ocorra e depois utilizam algum mecanismo para recuperar o sistema.

O protocolo CMB pode prevenir o *deadlock* através da utilização de mensagens nulas ou da transmissão de mensagens nulas sob demanda. Uma outra abordagem consiste em não prevenir, mas detectar e recuperar os processos em *deadlock*.

Uma técnica importante utilizada nos protocolos conservativos é o *lookahead*. *Lookahead* pode ser definido como o intervalo de tempo em que um processo pode garantir o que acontecerá no futuro com absoluta certeza (Misra, 1986) ou a capacidade de prever o que acontecerá, ou o que não acontecerá, no tempo de simulação futuro (Fujimoto, 1990). Um *lookahead* de L representa que o processo pode garantir que nenhuma mensagem nova de evento será criada com *timestamp* menor do que $T + L$, onde T é o tempo de simulação do processo.

3.2.2.1 CMB (Prevenção do Deadlock através de Mensagens Nulas)

O protocolo conservativo CMB requer que todos os canais de comunicação entre os processos lógicos sejam estabelecidos estaticamente. Para determinar quando é seguro processar um evento as mensagens devem chegar aos canais em ordem crescente de *timestamp*. Isto garante que nenhum outro evento que chegar no canal terá um *timestamp* menor do que o que está na cabeça da fila.

As mensagens que chegam em cada canal de entrada são armazenadas em uma fila FIFO, seguindo a restrição acima. Cada canal tem um relógio associado que possui: (a) o valor do *timestamp* da mensagem que está na cabeça da fila caso esta possua algum evento; (b) o valor da última mensagem caso a fila esteja vazia; (c) zero caso nenhuma mensagem

tenha chegado.

O relógio do processo lógico (*LVT – Local Virtual Time*) possui o menor valor entre os relógios dos canais de entrada. O processo seleciona repetidamente o canal com menor *timestamp* e caso haja uma mensagem, processa-a. Caso a fila esteja vazia o processo é bloqueado pois uma mensagem com um *timestamp* menor do que os dos outros canais pode chegar por este canal.

A figura 3.3 apresenta um processo lógico (*PL*) com três canais de entrada (*C1*, *C2* e *C3*) com seus respectivos relógios (2, 5 e 11). O menor valor do relógio é o do canal *C2*. Portanto, o *LVT* do processo *PL* é 2 e isto garante que *PL* pode com certeza seguir sua simulação até o tempo 2.

Figura 3.3. *LVT* de um processo lógico

O valor do relógio da simulação (*T*) é o menor valor dentre os relógios de todos os processos lógicos (*LVT*). Pode-se declarar que em um determinado ponto da simulação, o sistema físico tenha sido simulado até o tempo indicado por *T*, mesmo que algum processo lógico tenha simulado seu correspondente processo físico até um tempo maior que *T* (Misra, 1986).

Caso ocorra um ciclo de filas vazias, o sistema entra em *deadlock*, como pode ser observado na figura 3.4. Neste caso, cada *PL* está esperando na fila que contém o menor valor do relógio porque a fila está vazia.

Figura 3.4. Situação de *Deadlock*

Misra (1986) define que um conjunto D de processos encontra-se em *deadlock* se todas as seguintes condições forem satisfeitas:

1. Todo PL , pertencente a D , está esperando uma mensagem ou encerrou sua execução;
2. No mínimo um PL , pertencente a D , está esperando uma mensagem;
3. Um PL_i pertencente a D está esperando uma mensagem de algum PL_j , também pertencente a D , e não existe nenhuma mensagem de PL_j para PL_i .

Essas condições garantem que nenhum processo pertencente a D irá executar qualquer tipo de processamento já que cada um está esperando pelo outro.

Com o objetivo de impedir que o sistema entre em *deadlock*, foi desenvolvido um mecanismo que se baseia na transmissão de mensagens nulas, que será enviada quando não há nenhuma mensagem real a ser enviada.

A mensagem nula é utilizada somente para efeitos de sincronização e não corresponde a nenhuma atividade real do sistema. Uma mensagem E_{null} com *timestamp* T_{null} enviada de PL_i para PL_j indica que PL_i não enviará qualquer outra mensagem para PL_j antes de T_{null} . (Fujimoto, 1990; Ayani, 1993). As mensagens nulas são tratadas como qualquer outra mensagem, ou seja, o processo lógico atualiza as variáveis internas, o *LVT* e (eventualmente) gera novas mensagens (Misra, 1986).

O processo determina o *timestamp* de suas mensagens nulas através do valor do relógio de cada canal de entrada do processo lógico. Quando um LP termina de tratar um evento, envia uma mensagem nula para cada canal de saída com o valor mínimo entre os relógios de seus canais.

Este mecanismo impede o *deadlock*, mas fica claro o *overhead* causado pelas mensagens nulas. A simplicidade é o ponto mais atrativo deste mecanismo. Uma variação deste mecanismo é o envio de mensagens nulas sob demanda.

3.2.2.2 Transmissão de Mensagens Nulas sob Demanda

Neste mecanismo, a transmissão das mensagens nulas é feita sob demanda e não após cada evento, ou seja, as mensagens nulas são requisitadas (Chandy & Misra, 1981; Fujimoto, 1990).

As mensagens podem ser requisitadas por *timeout* ou quando o menor valor do relógio de um processo lógico for uma fila vazia (bloqueado). Quando isso ocorre, a próxima mensagem (nula ou não) é solicitada e o processo aguarda a resposta para continuar a execução.

A figura 3.5 ilustra a transmissão de mensagens nulas sob demanda. Em 3.5 (a) o processo PL_1 requisita para que PL_2 aumente o valor do relógio que conecta os dois processos. Este avanço só é possível uma vez que o valor do canal entre PL_1 e PL_2 é menor do

que o *LVT* de PL_2 . PL_2 envia uma mensagem nula para PL_1 com *timestamp* 12 e então PL_1 pode atualizar seu *LVT* para 9.

No caso da figura 3.5 (b) o avanço não foi possível pois o valor do canal é igual ao valor do *LVT* de PL_2 . PL_2 tem que primeiro avançar seu próprio *LVT* fazendo o mesmo pedido a todos os que enviam mensagens para PL_2 . Somente quando o valor do *LVT* de PL_2 for maior que o valor do canal (LP_2, LP_1) é que PL_2 pode enviar uma mensagem para PL_1 .

Figura 3.5. Transmissão de Mensagens Nulas sob Demanda

Os pedidos de avanço de relógio podem formar um ciclo, gerando *deadlock* (figura 3.6). Uma mensagem solicitando a avanço do relógio é enviada por Y para X, propagada para Z e então retorna para Y, que detecta o *deadlock* (Misra, 1986).

Figura 3.6. *Deadlock* no pedido de avanço de relógio

Misra (1986) afirma que a resolução do *deadlock* é difícil e pode ser feita através da determinação do mínimo dos “tempos dos próximos eventos”. Para cada processo pertencente ao conjunto de processos em *deadlock* o tempo em que uma mensagem será enviada é

determinado e o valor do relógio do processo com menor relógio é avançado para esse tempo.

A vantagem desse mecanismo é que a quantidade de mensagens nulas é bem menor do que no protocolo original (CMB). No entanto, quando ocorre o *deadlock*, o tempo para receber uma mensagem nula é maior uma vez que é necessária a transmissão de duas mensagens (uma requisitando o avanço do relógio e outra confirmando).

3.2.2.3 *Detecção e Recuperação do Deadlock*

Este mecanismo difere dos outros pelo fato de que não existe a transmissão de mensagens nulas, sendo que o *deadlock* pode acontecer. Um mecanismo é então responsável por detectar o *deadlock* e outro por quebrá-lo.

A detecção é feita através de um marcador que circula continuamente pela rede, devendo percorrer todos os canais da rede em algum tempo durante o ciclo. Este marcador é um tipo especial de mensagem e assim que o processo o recebe, envia-o através de sua rota depois de um certo tempo em que o processo esteja ocioso (não tenha mais mensagens para enviar).

O marcador carrega algumas informações para detecção do *deadlock*. Cada *PL* tem um *flag* de 1 bit que indica se o *PL* recebeu ou enviou alguma mensagem desde a última vez que o marcador passou por ele. Um *PL* é branco se não recebeu nem enviou nenhuma mensagem e preto caso contrário. Inicialmente, todos os *PL* são pretos. O marcador detecta o *deadlock* quando os últimos *N PL*'s que visitou forem brancos, sendo que *N* é o número de canais na rede.

Além de armazenar a quantidade de *PL*'s brancos que visitou, o marcador armazena também os tempos dos próximos eventos dos processos brancos. Quando o marcador detecta o *deadlock*, ele sabe o tempo do próximo evento e o *PL* onde este próximo evento deve ocorrer.

Uma desvantagem deste mecanismo é a sobrecarga de comunicação causada pela transmissão do marcador além da necessidade de um mecanismo para resolução do *deadlock* (Misra, 1986).

3.2.3 **Protocolos Otimistas**

Os protocolos otimistas, ao contrário dos conservativos, não impedem os erros de causa e efeito, não precisando 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*). A vantagem destes mecanismos é a possibilidade de explorar completamente o paralelismo em situações onde erros poderiam ocorrer, mas não ocorrem.

O mecanismo *Time Warp* é o mais conhecido protocolo otimista. Foi proposto por Jefferson em (Jefferson, 1985) e baseia-se no paradigma de *Virtual Time*.

3.2.3.1 Time Warp

O mecanismo *Time Warp* permite que um *PL* execute eventos em seu tempo de simulação local, denominado *Local Virtual Time (LVT)*, assim que exista uma mensagem na sua lista de entrada (Ayani, 1993). Deste modo, todos os eventos são considerados seguros e são executados.

Ele possui dois mecanismos principais: o mecanismo de controle local, que garante que os eventos são processados em ordem correta e é implementado independentemente em cada processador; e o mecanismo de controle global, que é responsável pelo gerenciamento de memória e pelo cálculo do *Global Virtual Time (GVT)* (Jefferson, 1985).

Para o mecanismo local garantir que os eventos sejam processados em ordem correta ele utiliza as seguintes estruturas para definir um processo lógico (figura 3.7) (Jefferson, 1985):

- **Nome do processo:** deve ser único no sistema;
- **Estado:** variáveis locais ao processo, tais como pilha de execução, contador de programa, etc.
- **Relógio virtual local:** determina o *LVT* do processo. O valor desse relógio não muda durante a ocorrência de um evento mas entre os eventos;
- **Lista de entrada de mensagens:** as mensagens que chegam são armazenadas nesta lista em ordem crescente de *timestamp* e são denominadas mensagens positivas. Algumas dessas mensagens já foram processadas mas continuam a fazer parte da lista para o caso de ser necessário realizar o *rollback*;
- **Lista de saída de mensagens:** esta lista contém uma cópia de todas as mensagens já enviadas pelo processo. Todas as mensagens têm *timestamp* menor do que o *LVT* e são denominadas mensagens negativas ou antimensagens. São utilizadas em caso de *rollback* caso seja necessário reenvia-las;
- **Lista de estados:** os estados recentes dos processos são armazenados nesta lista para viabilizar o *rollback*.

No mecanismo *Time Warp*, a mensagem com menor *timestamp* ainda não processada é sempre segura para ser processada. O menor *timestamp (LVT)* entre todas as mensagens de eventos enviadas e ainda não processadas é denominado *Global Virtual Time (GVT)*.

As mensagens são recebidas e processadas pelos processos sendo que a recepção de uma mensagem com *timestamp* menor do que uma mensagem já processada faz acionar o mecanismo de *rollback* (é detectado um erro de causa e efeito), como pode ser observado na figura 3.8. A mensagem que ativa o *rollback* é denominada *straggler*.

Figura 3.7. Estrutura do processo lógico A no Time Warp

Figura 3.8. Chegada de uma mensagem *straggler*

O mecanismo de *rollback* faz a simulação voltar a um estado seguro, anterior ao da violação (Fujimoto, 1990) e esta é a maneira pela qual o mecanismo se desfaz de seus processamentos errôneos. Para isso é necessário que os estados dos processos sejam periodicamente salvos na lista de estados. As mensagens que são enviadas, por exemplo, de um processo A para um processo B, são salvas na lista de entrada do processo B e na lista de

saída do processo A.

Quando um erro de causa e efeito é detectado, a mensagem que está na lista de saída (mensagens negativas ou antimensagens) é enviada novamente, fazendo com que o processo receptor tenha duas mensagens iguais e de sinais opostos (positiva e negativa) em sua lista de entrada. Estas mensagens são então eliminadas (Jefferson, 1985). Caso o processo já tenha executado esta mensagem, outras antimensagens são enviadas recursivamente.

O mecanismo global é responsável pelo cálculo do *GVT* e também pelo gerenciamento da memória, uma vez que a quantidade de memória necessária para o desenvolvimento deste mecanismo é maior do que a utilizada pelo mecanismo CMB ou mesmo por uma simulação sequencial.

O *GVT* consiste no menor valor entre todos os *LVT*'s dos processos lógicos. Após o cálculo do *GVT* todas as informações referentes a estados anteriores a esse tempo podem ser descartadas, uma vez que é garantido que até este tempo não é possível ocorrer qualquer erro de causa e efeito (Jefferson, 1985).

- ***Estratégias de cancelamento***

Após a ocorrência de um *rollback*, algumas mensagens enviadas para outros processos precisam ser canceladas. Para tanto, três estratégias de cancelamento foram desenvolvidas: cancelamento agressivo, cancelamento preguiçoso e cancelamento dinâmico (Fujimoto, 1990).

- **Cancelamento agressivo:** as antimensagens são enviadas imediatamente após a detecção do erro de causa e efeito. As antimensagens enviadas podem causar uma cascata de *rollbacks*;
- **Cancelamento preguiçoso:** as antimensagens não são enviadas imediatamente após a detecção do erro de causa e efeito. Elas esperam para ver se a nova execução não gera as mesmas mensagens, evitando enviar antimensagens desnecessárias. Dependendo do tipo de aplicação, o cancelamento preguiçoso pode melhorar ou piorar o desempenho do sistema.
- **Cancelamento dinâmico:** cada PL decide qual estratégia de cancelamento (agressivo ou preguiçoso) vai utilizar.

- ***Estratégias de armazenamento das variáveis de estado***

O armazenamento das variáveis de estado é necessário para garantir que cada estado que precise ser recuperado em caso de *rollbacks* possa ser gerado novamente. Vários métodos têm sido propostos e o mais simples é o *copy state saving*, o qual armazena todos os estados toda vez que um evento é executado. Esta estratégia leva a uma alta sobrecarga de tempo e

espaço de armazenamento. De forma a diminuir esta sobrecarga, outras estratégias foram propostas e estas se classificam em duas categorias: (a) *sparse state saving* e (b) *incremental state saving*. Na categoria *sparse state saving* os estados são armazenados por completo mas não toda vez que um evento é executado e sim, periodicamente. Quanto acontece um *rollback* as informações dos estados armazenados por último são recuperados. Na categoria *incremental state saving* somente a parte dos estados que sofreram alteração é que são atualizadas (Rönngren *et al.*, 1996, Fujimoto, 2000).

3.2.4 Comparação entre os protocolos conservativos e otimistas

Uma das formas de se realizar a comparação entre os protocolos conservativos e otimistas é através da avaliação de desempenho dos mesmos.

O desempenho dos protocolos conservativos e otimistas tem sido bastante estudado e é difícil determinar regras de quando deve ser utilizado um ou o outro protocolo. Estudos como os desenvolvidos por: Song (2001), Lim *et al.* (2000), Bagrodia & Takai (2000), Ulson (1999) e Miguel *et al.* (1998) analisam o desempenho dos protocolos conservativos, enquanto os estudos desenvolvidos por Spolon (2001), Quaglia *et al.* (1999) e Ferscha & Johnson (1996) analisam o desempenho dos protocolos otimistas. Já os trabalhos de Balakrishnan *et al.* (2001), Ferscha *et al.* (2001), Reynolds *et al.* (1989) e Cavitt *et al.* (1996) propõem mecanismos para análise dos protocolos de sincronização, considerando tanto os protocolos conservativos como os otimistas.

Fazendo uma análise que tem como foco o funcionamento, mais especificamente a definição dos protocolos, pode-se dizer que a maior desvantagem dos protocolos conservativos é o fato de o paralelismo não poder ser totalmente explorado, uma vez que quando um evento E_1 pode, direta ou indiretamente, afetar E_2 , estes são executados sequencialmente. Outra desvantagem é a necessidade dos canais serem especificados estaticamente, ou seja, os processos e os canais de comunicação não podem ser especificados durante a execução da simulação. O desempenho dos protocolos conservativos está também diretamente ligado à capacidade de “olhar à frente” no tempo de simulação futuro. Eles não são aconselháveis para sistemas que têm pouca capacidade de *lookahead*, mesmo que apresentem um alto grau de paralelismo. (Fujimoto, 2000). Alguns trabalhos encontrados na literatura estudam a importância e a influência do *lookahead* nos protocolos conservativos (Peterson & Chamberlain, 1993; Janneck, 1998) e outros mostram como calcular esse valor (Nicol, 1988; Wagner & Lazowska, 1989).

Já os protocolos otimistas oferecem grande potencial como mecanismos para simulação de propósito geral (Overeinder *et al.*, 1991). A desvantagem é a necessidade de salvar periodicamente o estado dos processos lógicos, utilizando mais memória do que os protocolos conservativos. Isso faz com que este mecanismo seja mais difícil de ser implementado

(Fujimoto, 2000). As vantagens são que ele normalmente apresenta um *speedup* maior do que o apresentado pelos protocolos conservativos e também não é necessário que a topologia dos processos seja definida estaticamente. Diversas otimizações dos protocolos otimistas têm sido propostas com o objetivo de melhorar a utilização da memória e também limitar seu otimismo e reduzir o número de *rollbacks* (Steinman, 1992; Steinman, 1993; Das, 1996).

A tabela 3.1 ilustra a comparação de algumas características dos protocolos conservativos e otimistas (Ferscha, 1995; Vee & Hsu, 1999; Fujimoto, 2000).

A análise realizada acima é importante, porém não suficiente para determinar a escolha de uma das abordagens (conservativa ou otimista). O desempenho dos protocolos de sincronização deve levar em consideração outros dois fatores: a plataforma onde a simulação vai ser realizada e o modelo a ser simulado. A relação entre esses fatores é definida em (Ferscha *et al.*, 2001) e pode ser observada na figura 3.9.

Tabela 3.1 Comparação entre os protocolos conservativo e otimista (Fujimoto, 2000)

	Conservativo	Otimista
Paralelismo	Limitado pelo pior caso	Sem limitações
Reduz o desempenho	Baixo <i>lookahead</i>	Salvar estados / sobrecarga <i>rollback</i>
Utilização da memória	Não é necessário <i>logs</i>	Precisa de <i>logs</i> dos estados e mensagens
Configuração dos PLs	Estática	Dinâmica
Implementação do protocolo	Fácil	Mais difícil. Deve-se considerar os <i>rollbacks</i>
Desenvolvimento de aplicações	Que exploram o <i>lookahead</i>	Mais robusto a mudanças no modelo Mais transparência no protocolo

Figura 3.9. Interação entre fatores que influenciam o desempenho dos protocolos de sincronização

A plataforma onde será executada a simulação afeta o desempenho como um todo pois ambos os protocolos são influenciados pelo poder computacional, pela topologia e capacidade de comunicação da rede e pelo sistema operacional que está sendo utilizado. Pode-se dividir esses fatores em:

- *Hardware*: velocidade do processador e número de processadores, memória;
- Comunicação: topologia da rede, latência, mecanismo de comunicação (PVM, MPI, etc.);
- *Software*: sistema operacional, linguagem (C, Java, etc.), otimizações do compilador;

O modelo a ser simulado afeta o desempenho no sentido de que dependendo de alguns fatores, tais como particionamento e grau de paralelismo, granulosidade, tamanho do período transiente e do nível de correlação entre as observações coletadas, os protocolos podem oferecer um desempenho melhor ou não.

O fator estratégia de simulação já foi discutido no início da seção e pode-se dizer que a influência no desempenho da simulação se deve principalmente a dois tópicos:

- Mecanismo de simulação: gerenciamento da lista de eventos futuros, modelo de gerenciamento do tempo, escalonamento;
- Protocolo de sincronização: fatores relacionados aos protocolos, tais como: *lookahead* e tratamento do *deadlock* (conservativo), cálculo do GVT, estratégias de armazenamento das variáveis de estado e coleta de fóssil (otimista);

Com base nos estudos citados nesta seção pode-se concluir que nenhuma abordagem irá fornecer um bom desempenho para todos (ou quase todos) os tipos de aplicações. Uma abordagem que é considerada boa para uma aplicação pode não se boa para outra (Vee & Hsu, 1999). Uma análise mais detalhada do desempenho é desenvolvida no capítulo 5, onde um módulo do ambiente proposto nesta tese é detalhado (Módulo Avaliador).

Analisada a simulação SRIP e seus protocolos de sincronização, a simulação MRIP é o foco da próxima seção.

3.3 *Multiple Replications in Parallel* – MRIP

A técnica *Multiple Replications in Parallel* (MRIP) é uma alternativa quando se deseja utilizar diversos processadores para realizar uma simulação. Ao contrário das técnicas utilizadas na SRIP, onde um único programa de simulação é dividido em processos lógicos e cada processo é simulado em um processador, na técnica MRIP várias instâncias (denominadas a partir daqui como replicações) de um mesmo programa de simulação sequencial são executadas independentemente em diferentes processadores (Glynn &

Heidelberger, 1992; Ewing et al., 1998). Deste modo, primeiramente os dados do período transiente devem ser eliminados em cada replicação e, após atingir o período de estimação dos parâmetros, os resultados produzidos por cada replicação começam a contribuir na análise da variável estimada. Esses resultados são enviados para um analisador global que calcula o intervalo de confiança da(s) variável(is) estimada(s) e quando esta(s) atinge(m) a precisão requerida, a simulação é encerrada (Ewing et al., 1997).

Devido ao fato de que cada replicação é a execução de uma simulação seqüencial, esta deve utilizar algum método para cálculo de intervalo de confiança. O trabalho de Edjair Mota (Mota et al., 1999; Mota et al., 2000) analisa a utilização do método média dos lotes sobrepostos (OBM – *Overlapping Batch Means*) em simulações MRIP.

Ao contrário da *SRIP*, *MRIP* pode ser amplamente utilizada em qualquer sistema, independente do sistema possuir paralelismo ou não. As replicações só não são aplicadas quando (Glynn & Heidelberger, 1992):

1. O modelo é grande o suficiente de modo que uma replicação não pode ser executada em um único processador, em um tempo razoável;
2. A variância de cada replicação é muito pequena de modo que os resultados são quase determinísticos. Neste caso, replicar é um desperdício de processamento.

Apesar do método ser conceitualmente simples, alguns cuidados devem ser tomados em relação ao número de processadores (replicações), ao tamanho das replicações e à quantidade de observações a serem eliminadas no período transiente de modo a gerar um intervalo de confiança dos parâmetros observados válido (Glynn & Heidelberger, 1992).

Todo processo de estimação tem um pouco de variabilidade e esta variabilidade é normalmente medida pela variância do estimador (normalmente, estimadores da média e da variância). Heidelberger (1988) provou que as *bias* (outra componente que mede a variabilidade) dos estimadores mais comuns aumentam conforme aumenta o número de processadores. Desse modo, quando esses métodos são utilizados para analisar resultados paralelos seus desempenhos precisam ser avaliados, pois, devido ao efeito da paralelização, a combinação linear dos valores estimados tende a convergir para o valor errado (Heidelberger, 1988).

As vantagens na utilização da *MRIP* são (Ewing et al., 1998):

- Pode ser aplicado a qualquer programa de simulação, sem a necessidade de paralelizá-lo ou modificá-lo;
- Desde que as simulações são independentes, se n replicações estão sendo executadas em n processadores, n resultados serão produzidos enquanto na versão seqüencial somente um seria produzido, ou seja, produz um *speedup* aproximadamente igual ao número de processadores;

- Tolerância à falha, uma vez que são várias instâncias de um mesmo programa sendo executadas. Um cuidado especial deve ser tomado quanto ao analisador global pois se ocorre algum problema neste analisador, a simulação pára.

O ambiente *Akaroa-2* (Ewing *et al.*, 1998; Ewing *et al.*, 1999) é um exemplo de ambiente que utiliza a técnica *MRIP*, e é descrito na seção 3.6.

3.4 Comparação entre *SRIP* e *MRIP*

Heidelberg em (Heidelberg, 1986) estabeleceu uma comparação entre as abordagens *SRIP* e *MRIP*. Um modelo foi formulado e algumas conclusões foram obtidas de modo a determinar qual abordagem é estatisticamente mais eficiente. O modelo desenvolvido compara o erro médio quadrado gerado pelas duas abordagens após um tempo fixo de processamento.

As principais métricas utilizadas para medir o desempenho em computação paralela são: *speedup* e eficiência. Quando *SRIP* é considerada, a eficiência é determinada por outros fatores, incluindo (Heidelberg, 1986):

1. O mecanismo utilizado para sincronização dos eventos;
2. A sobrecarga na comunicação entre os processos e entre os processadores;
3. A arquitetura de *hardware* e *software* disponíveis;
4. As características do sistema simulado.

O resultado produzido pela análise de Heidelberg (1986) conclui que outros fatores devem ser levados em consideração: período transiente e a variabilidade do sistema a ser simulado, a quantidade de processamento disponível e o número de processadores.

As conclusões obtidas a partir do modelo formulado foram (Heidelberg, 1986):

- Se o tamanho da simulação é longo ou se o período transiente é fraco (as observações iniciais devem influenciar pouco as observações subsequentes e consequentemente não é necessário eliminar muitas observações) então *MRIP* é estatisticamente mais eficiente do que *SRIP*. Esta situação pode ser observada na figura 3.10. Devido ao fato do período transiente ser pequeno e o tamanho da simulação longo, a soma de todas as observações eliminadas não representa uma grande perda;
- Se o tamanho da simulação é pequeno, o período transiente é forte (as observações iniciais devem influenciar muito as observações subsequentes e consequentemente é necessário eliminar muitas observações) ou existe um grande número de processadores P disponíveis ($speedup \propto P \rightarrow \infty$ conforme $P \rightarrow \infty$) então a *SRIP* é estatisticamente mais eficiente do que *MRIP* (figura 3.11). A proporção do número

de observações que precisam ser eliminadas em relação ao número total de observações coletadas durante a simulação inviabiliza a utilização de *MRIP*. A análise dos resultados pode, neste caso, ser feita utilizando o método *Batch Means* já que neste método o período transiente só acontece uma vez.

- Se o número de processadores é grande e o período transiente é relativamente moderado em relação ao tamanho da simulação então uma abordagem que combina *SRIP* e *MRIP* é aconselhada. Supondo que existam disponíveis P processadores e R é o número de replicações, se cada replicação utiliza *SRIP* em M processadores então $P = R * M$. Neste caso é melhor utilizar um valor pequeno para M (poucos processadores por replicação) e um valor grande para R (várias replicações).

Figura 3.10. Utilização da abordagem *MRIP*

Figura 3.11. Utilização da abordagem *SRIP*

Independente da abordagem de simulação distribuída escolhida (SRIP ou MRIP), o desenvolvimento de uma simulação sofre algumas alterações em seu diagrama. Essas alterações são discutidas na próxima seção.

3.5 Fases de desenvolvimento para uma simulação distribuída

Como pôde ser observado na especificação das simulações SRIP e MRIP, o desenvolvimento de uma simulação distribuída exige algumas alterações nas fases de desenvolvimento de uma simulação. Trata-se da inclusão de duas fases e a figura 3.12 ilustra a nova seqüência. As duas novas fases estão representadas na figura por linhas pontilhadas.

Figura 3.12. Fases de desenvolvimento de uma simulação distribuída

Particionamento

No caso da simulação SRIP, deve-se nesta fase decidir qual é o particionamento dos centros de serviço (no caso de um modelo em redes de filas) e dos *templates* centro de serviço (no caso de um modelo em *queueing statecharts*) nos processos lógicos que apresenta o melhor desempenho. Esta fase torna-se desnecessária para a simulação MRIP.

Configuração da arquitetura

Para executar uma simulação distribuída é necessário que seja estabelecida a arquitetura em que a simulação será executada, e isto é válido tanto para a simulação SRIP quanto para a simulação MRIP. Como descrito no Apêndice A sobre Computação Paralela, as possíveis arquiteturas utilizadas na execução de simulação distribuída são: multiprocessador, multicomputador e os computadores distribuídos. No caso da utilização dos computadores distribuídos existe a possibilidade da utilização de ambientes de passagem de mensagem para a comunicação entre os processos. Neste caso, é necessário estabelecer quais máquinas farão parte da máquina virtual.

3.6 Recursos disponíveis para o desenvolvimento de simulações distribuídas

As vantagens proporcionadas pela simulação distribuída não podem ser totalmente utilizadas caso o usuário tenha que se preocupar com todos os detalhes de comunicação e sincronização. Visando solucionar este problema, o usuário pode optar pela utilização de linguagens ou bibliotecas (extensões funcionais) ou ainda pela utilização de ambientes automáticos. As linguagens e bibliotecas têm sido desenvolvidas de modo a liberar o usuário dos detalhes de sincronização e permitir que concentre seus esforços na modelagem do sistema. (Low *et al.*, 1999). Já os ambientes automáticos possuem o objetivo de não somente liberar o usuário dos detalhes de comunicação e sincronização como também liberar da tarefa de transcrição do modelo em um programa de simulação. A seguir estes dois tópicos são detalhados.

3.6.1 Linguagens e Bibliotecas

As linguagens para simulação SRIP devem fornecer ao usuário um conjunto de primitivas de modo a permitir a especificação do modelo de simulação construído. As seguintes linguagens foram desenvolvidas com este objetivo (Low *et al.*, 1999):

- APOSTLE (*A Parallel Object-oriented SimlaTion Language*): desenvolvido por DERA (*Defence Evaluation and Research Agency*), é uma versão orientada a objetos de um protocolo otimista;

- Maisie: desenvolvido na Universidade da Califórnia – USA (UCLA) implementa diversos protocolos de sincronização, incluindo protocolos otimistas e conservativos;
- Moose (Maisie-based Object-Oriented Simulation Environment): desenvolvida na UCLA, é uma versão orientada a objetos da linguagem Maisie;
- Parsec (Parallel Simulation Environment for Complex Systems): desenvolvido na UCLA e é uma evolução da linguagem Maisie;
- ModSim: desenvolvido inicialmente desenvolvido por CACI Products, Inc. e após 1990 por Jade Simulations, é uma linguagem orientada a objetos baseada na linguagem Modula-2 e utiliza o protocolo *Time Warp* da Jade;
- YADDES: desenvolvido na Universidade de Waterloo, Canadá, é uma linguagem de especificação que também implementa diversos protocolos de sincronização.

A tabela 3.2 compara algumas características dessas linguagens.

Tabela 3.2. Comparação entre diversas linguagens para SRIP (Low et al., 1999)

Linguagens Características	APOSTLE	Maise/Parsec	Moose	ModSim	YADDES
Protocolo	Otimista	Vários	Vários	Otimista	Vários
Orientação	Processo / Evento	Processo / Evento	Processo / Evento	Processo / Evento	Processo
Paradigma de Programação	Orientada Objeto	Estruturada	Orientada Objeto	Orientada Objeto	Especificação
Mapeamento	Manual	Manual	Manual	Automático	Manual
Visualização	Não	MVPE	MVPE	Não	Não
Depuração	Não	Nível código	Nível código	Não	Não
Estatísticas	Não	Sim	Sim	Sim	Sim

As bibliotecas para simulação SRIP fornecem ao usuário um conjunto de rotinas as quais devem ser explicitamente referenciadas. Se nas linguagens para SRIP as características do protocolo de sincronização são escondidas do usuário, nas bibliotecas isso não acontece. Para usuários mais experientes, essa é a vantagem de se utilizar as bibliotecas e não as linguagens. Algumas bibliotecas para SRIP podem ser citadas, tais como (Low *et al.*, 1999):

- CPSim: único ambiente comercializado, foi desenvolvido pelo Dr. Bojan Groselj e comercializado por Boyan Tech, Inc. É uma biblioteca da linguagem C e utiliza protocolo conservativo;

- GTW e TWOS: bibliotecas da linguagem C, implementam o protocolo *Time Warp*. GTW foi desenvolvido pela Georgia Institute of Technology e TWOS foi desenvolvido pelo JTL (Jet Propulsion Laboratory);
- ParaSol, PSK (Parallel Simulation Kernel), SimKit, SPaDES (Structured Parallel Discrete-Event Simulation), SPEEDES e WARPED: bibliotecas orientadas a objeto utilizando a linguagem C++ e implementam protocolos otimistas. Os desenvolvedores são: ParaSol: Universidade de Purdue; PSK: Royal Institute of Technology, Suécia; SPaDES: Universidade Nacional de Singapura; SPEEDES: Jet Propulsion Laboratory; WARPED: Universidade de Cincinnati.
- ParSMPL (Ulson, 1999): desenvolvido no Grupo de Sistemas Distribuídos e Programação Concorrente do ICMC-USP, é um ambiente baseado na extensão funcional SMPL (linguagem C) e utiliza o protocolo conservativo CMB;
- JWarp (Bizarro *et al.*, 1998): desenvolvido na Universidade de Coimbra, utiliza a linguagem Java e implementa o protocolo *Time Warp*;
- JTED (Cowie, 1998): desenvolvido pela Cooperating Systems Corporation utiliza a linguagem Java e implementa o protocolo conservativo.

Algumas dessas bibliotecas foram desenvolvidas com o objetivo de realizar pesquisas em relação aos protocolos. Por exemplo, PSK foi desenvolvido com o objetivo de avaliar o desempenho de diferentes estratégias de salvamento dos estados no *Time Warp* (Low *et al.*, 1999). ParSMPL foi desenvolvido com o objetivo de avaliar o desempenho de simulações distribuídas utilizando o protocolo CMB em diferentes plataformas computacionais (Ulson, 1999). A tabela 3.3 compara algumas características dessas bibliotecas (Low *et al.*, 1999).

Tabela 3.3. Comparação entre diversas bibliotecas para SRIP

Bibliotecas Características	CPSim	GTW	ParaSol	PSK
Protocolo	Conservativo	Otimista Seqüencial	Otimista	Otimista
Orientação	Evento	Evento	Processo	Evento
Paradigma de Programação	Gabarito	Gabarito	Orientação objeto	Orientação objeto
Mapeamento	Manual e Automático	Manual	Manual	Automático
Visualização	Não	Sim	Não	Sim
Depuração	Sim	Sim	Sim	Sim
Estatísticas	Sim	Sim	Não	Não

Bibliotecas Características	Simkit	SPaDES	SPEEDES	TWOS
Protocolo	Otimista Sequencial	Otimista	Otimista	Otimista
Orientação	Evento	Processo	Evento	Evento
Paradigma de Programação	Orientação objeto	Orientação objeto	Orientação objeto	Gabarito
Mapeamento	Manual	Manual	Manual	Manual
Visualização	Não	Não	Não	Sim
Depuração	Sim	Não	Sim	Sim
Estatísticas	Não	Não	Não	Sim

Bibliotecas Características	WARPED	ParSMPL	Jwarp	JTED
Protocolo	Otimista	Conservativo	Otimista	Conservativo
Orientação	Evento	Evento	Evento	Evento
Paradigma de Programação	Orientação objeto	Estruturada	Orientação objeto	Orientação objeto
Mapeamento	Manual	Manual	Não	Não
Visualização	Não	Não	Sim	Sim
Depuração	Sim	Não	Não	Não
Estatísticas	Não	Não	Não	Não

3.6.2 Ambientes de Simulação Automáticos

Um obstáculo na elaboração de uma simulação é a transcrição do modelo definido pelo usuário em um programa de simulação. Para ajudar o usuário neste processo de transcrição, foi desenvolvido o conceito de ambiente de simulação. Esses ambientes são ferramentas que oferecem ao usuário uma série de recursos que visam facilitar o desenvolvimento de uma simulação. Por exemplo, o programa de simulação pode ser gerado e executado automaticamente a partir de uma representação gráfica do modelo do sistema a ser avaliado. Para isto é necessário a inclusão de editores gráficos, interpretadores, compiladores e otimizadores (Tanir & Sevinc, 1994). O ambiente de simulação é definido então como um

ambiente de simulação automático.

Nos ambientes de simulação automáticos uma característica importante é a programação visual. Utilizando a programação visual, o modelo de simulação é criado graficamente na tela e os resultados (comportamento do sistema) são observados através de gráficos ou animações. A programação visual e a saída visual são características da simulação visual interativa (VIS – *Visual Interactive Simulation*) (Bell & O’Keefe, 1987).

Quando as informações são apresentadas graficamente, o usuário pode manipulá-las de um modo mais eficiente tornando a programação visual particularmente desejável e vantajosa.

Existem vários ambientes de simulação, sendo que a maioria são comerciais. A seguir serão descritos alguns ambientes (comerciais ou acadêmicos), os quais apresentam todas ou algumas das características descritas. Estes ambientes foram desenvolvidos para simulação sequencial e/ou distribuída.

3.6.2.1 ARENA

ARENA (Swain, 1995; Swet & Drake, 2001) é um ambiente de simulação comercial de propósito geral baseado na linguagem SIMAN V (SIMulation ANalysis). SIMAN é uma linguagem de simulação que modela sistemas discretos, contínuos ou híbridos. ARENA originou o uso de *templates* (módulos pré-definidos) de simulação. Com isto, o *software* é facilmente adaptado para uma indústria específica, uma companhia ou um projeto. Um Editor Profissional ARENA permite que o usuário projete seus próprios *templates*. O ARENA permite também a construção de modelos animados.

3.6.2.2 OMNeT++

OMNeT++ é um sistema de simulação discreta sequencial desenvolvido na área acadêmica com o objetivo de ser uma alternativa aos custosos produtos comerciais da área de simulação. Foi desenvolvido para simular principalmente redes de computadores, multiprocessadores e outros sistemas distribuídos, podendo ainda ser utilizado para simular outros tipos de sistemas (Varga, 2001). OMNeT possui os seguintes componentes:

- Biblioteca de simulação;
- Compilador para a linguagem de descrição de topologia – NED (*Network Description*);
- Editor gráfico de redes para os arquivos NED – GNED (*Graphical Network Editor*);
- Interface gráfica para a execução da simulação;
- Interface em linha de comando para a execução da simulação;

- Ferramenta para impressão dos gráficos de saída;
- Utilitários (gerados de números aleatórios, ferramenta para criação de *makefiles*, etc);
- Documentação, exemplos de simulação

A figura 3.13 ilustra a simulação de uma rede *token ring* e os resultados da simulação sendo apresentados através de um histograma.

Figura 3.13. Simulação de uma rede *token ring* no OMNeT++

3.6.2.3 ASiA

O ASiA (Ambiente de Simulação Automático) é um ambiente de simulação desenvolvido também na área acadêmica para simulação sequencial no qual o programa é automaticamente gerado pelo sistema, baseado em uma especificação gráfica do modelo fornecida pelo usuário (Santana, 1996; Santana 1996a).

Esse ambiente tem por objetivo afastar o usuário da tarefa de transcrição do modelo em um programa de simulação. O usuário fornece o modelo do sistema considerado e os parâmetros necessários para a simulação através de um editor gráfico. As demais tarefas serão executadas automaticamente pelo ASiA.

Quatro estágios básicos são utilizados para a obtenção dos resultados no sistema de simulação automático (figura 3.14)

- Editor gráfico: fornece ao usuário uma interface amigável, organizando e testando a consistência dos dados de entrada;
- Gerador de aplicações: responsável pela geração do programa de simulação;
- Estágio de saída e análise dos dados: responsável por analisar os resultados fornecidos pela simulação;

- Saída gráfica: responsável por apresentar os resultados de uma maneira gráfica.

Figura 3.14. Visão geral do ASiA

O editor gráfico fornece um ambiente que permite ao usuário inserir seu modelo e os parâmetros necessários para a simulação através de um ambiente gráfico (figura 3.15) (Spolon, 1994; Bruschi *et al.*, 1999).

Figura 3.15. Tela principal do ASiA com um modelo de redes de filas

A segunda fase do sistema de simulação automática não depende da interferência direta do usuário. As informações coletadas e organizadas pelo editor gráfico em estruturas de dados previamente definidas alimentam um gerador de aplicações que fornecerá como saída o programa de simulação. Esse programa é descrito de acordo com a linguagem de simulação

adotada (Spolon, 1994a). Na atual versão do ASiA, a ferramenta adotada é o SMPL (MacDougall, 1987). Após isso, o programa de simulação gerado é executado e os resultados são obtidos.

O último estágio do sistema de simulação automático oferece ao usuário a possibilidade da visualização gráfica dos resultados obtidos na simulação (Pedroso, 1996) e também uma análise estatística dos resultados (Oliveira, 1998).

Novos módulos foram adicionados ao ASiA com o objetivo de facilitar a construção de modelos que representem arquiteturas de computadores - Módulo Arquitetura de Computadores (Bruschi, 1997; Bruschi *et al.*, 1999) e redes de computadores - Módulo Redes de Computadores (Kawabata, 1999).

A extensão desenvolvida para simulação de arquiteturas de computadores permite que o usuário defina diferentes tipos de arquiteturas, utilizando recursos básicos, tais como: processadores, disco, memória, etc. Além disso, a extensão oferece quatro tipos de arquitetura, cujos modelos já estão definidos: arquitetura de von Neumann, arquitetura de von Neumann com *pipeline*, arquitetura MIMD com memória compartilhada e arquitetura MIMD com memória distribuída. (Bruschi, 1997).

A extensão desenvolvida para redes de computadores permite ao usuário a simulação de redes já definidas, tais como redes Ethernet e redes *Token-Ring* (com a possibilidade de alteração dos parâmetros de entrada), ou definir novas redes através da utilização dos recursos da barra de ferramentas (Kawabata, 1999).

3.6.2.4 *UCLA Simulation Environment*

O ambiente de simulação desenvolvido na Universidade da Califórnia (UCLA) é dirigido a simulação distribuída e possui três componentes: a linguagem de simulação paralela Parsec, uma interface, denominada Pave (PARsec Visual Environment) e os sistemas que implementam os algoritmos de simulação. Os principais objetivos deste ambiente são: fornecer uma fácil migração dos modelos de simulação nos sistemas operacionais; possibilitar sua utilização em plataformas com memória compartilhada ou distribuída, suportando um conjunto diverso de protocolos de sincronização e suportar modelagem visual ou hierárquica (Bagrodia *et al.*, 1998).

Os *front-ends* para programação que o sistema suporta são: linguagem de simulação Parsec; biblioteca Compose, escrita em C++ e Pave. A figura 3.16 ilustra o ambiente, incluindo o *hardware* no qual o ambiente pode ser executado, o ambiente para comunicação, sistema operacional, algoritmos de sincronização e interfaces dos programas (Bagrodia *et al.*, 1998).

A linguagem Parsec adota a abordagem de interação de processos. Um programa consiste de um conjunto de entidades e funções em C, sendo que cada entidade é um processo

lógico. Os eventos são modelados através de troca de mensagens entre as entidades.

Figura 3.16. Visão geral do UCLA Simulation Environment

3.6.2.5 *DISplay: A System for visual-interaction in distributed simulation*

Este ambiente objetiva fornecer uma biblioteca para interface gráfica independente da aplicação, podendo ser unida a qualquer *software* de simulação, sequencial ou paralelo (Mascarenhas *et al.*, 1995). As vantagens desse ambiente incluem: facilidade de uso (um conhecimento especializado sobre visualização está acoplado ao ambiente, facilitando a utilização por iniciantes), flexibilidade (o sistema pode ser utilizado para várias aplicações, incluindo simulações distribuídas/paralelas e sequenciais), reutilização de código (o código para interação gráfica pode ser utilizado em diferentes aplicações) e modularidade (o código para interação gráfica fica em uma camada acima da aplicação) (Mascarenhas *et al.*, 1995).

3.6.2.6 *Akaroa-2*

Akaroa-2 é um ambiente orientado a objetos e escrito em C++ que utiliza a técnica *MRIP*, possibilitando a execução paralela de programas de simulação (replicações). Os resultados estimados obtidos nas replicações alimentam um analisador global que automaticamente pára a simulação quando a precisão desejada é atingida (Ewing *et al.*, 1999).

Seus principais componentes são: *akmaster*, *akslaves*, *akrun* e os *simulation engines* (replicações) como pode ser observado na figura 3.17. Cada um dos componentes (caixa com bordas mais grossas) é um processo UNIX e estes se comunicam através do protocolo TCP/IP.

O processo *akmaster* é responsável por coordenar as atividades de todos os outros processos iniciados pelo Akaroa. Suas tarefas são: iniciar novas simulações, manter informações sobre os estados das simulações que estão sendo executadas, executar a análise global dos dados produzidos e decidir quando a simulação deve parar.

Figura 3.17. Arquitetura do Akaroa

O processo *akslave* está sempre sendo executado nas máquinas onde as replicações serão executadas. Sua principal função é iniciar as replicações em sua máquina. O programa *akrun* é utilizado para iniciar uma simulação.

Seguindo os princípios da simulação sequencial, cada replicação realiza uma análise nos seus próprios resultados para calcular o que é chamado de *local estimate*. Atualmente, o Akaroa implementa os métodos *Batch Means* e Análise Espectral. Regularmente, em determinados *checkpoints*, a replicação envia os seus valores estimados para o processo *akmaster*, onde os valores de todas as replicações são combinados em uma estimativa global, denominada *global estimate*. Sempre que uma estimativa global é calculada, um novo intervalo de confiança para um determinado nível de confiança determinado pelo usuário é avaliado. Esse valor é então comparado com o intervalo especificado pelo usuário e, se for satisfatório, a simulação é encerrada.

Desta forma, o Akaroa implementa o método das replicações utilizando diferentes processadores e os métodos *Batch Means* ou Análise Espectral dentro de cada replicação.

3.7 Considerações Finais

A motivação para o desenvolvimento de simulação distribuída é o tempo necessário que se deve executar um programa de simulação de modo a se obter resultados confiáveis. Tendo isso em vista, neste capítulo foram apresentadas duas abordagens para o desenvolvimento de uma simulação distribuída: a abordagem SRIP (*Single Replication in Parallel*) e a abordagem MRIP (*Multiple Replication in Parallel*), sendo discutidas as vantagens e desvantagens de

cada abordagem.

O principal problema da *SRIP* é a sincronização, pois, neste caso, o sistema é quebrado em processos e estes precisam estar sempre sincronizados, evitando os erros de causa e efeito. Duas classes de protocolos para sincronização foram estudadas: os conservativos e os otimistas. Por outro lado a *SRIP* apresenta vantagens quando o sistema possui um paralelismo intrínseco.

A técnica *MRIP* apresenta a vantagem de não precisar modificar (paralelizar) o programa de simulação uma vez que cada replicação é um programa seqüencial. Uma desvantagem é os cuidados que devem ser tomados na análise.

Para ambas as abordagens existem linguagens, bibliotecas e ambientes desenvolvidos especificamente de forma a facilitar o desenvolvimento da simulação.

Toda a teoria descrita neste capítulo é utilizada como base para o desenvolvimento do ASDA (Ambiente de Simulação Distribuída Automático), objetivo desta tese de doutorado. O ASDA tem como objetivo facilitar o desenvolvimento de simulações distribuídas, permitindo o desenvolvimento de simulações utilizando as abordagens *SRIP* e *MRIP* e é descrito no próximo capítulo.

ASDA – Um Ambiente de Simulação Distribuída Automático

4.1 Considerações Iniciais

Como pode ser observado nos capítulos 2 e 3, o desenvolvimento de uma simulação não é uma tarefa trivial. Para garantir a validade dos resultados obtidos, uma série de conceitos de diferentes áreas devem ser empregados. O usuário de simulação deve ser capaz de aplicar todos os passos estipulados no algoritmo definido na seção 2.2, isto é, o usuário deverá conhecer: 1) técnicas de modelagem de um sistema (como abstrair as características essenciais ao modelo, como representá-las, etc.); 2) técnicas de validação e verificação de um modelo e de uma simulação; 3) análise estatística de resultados; 4) utilização de ferramentas de simulação; etc.

Um problema adicional na utilização de simulação é o longo tempo de execução necessário para obtenção de resultados confiáveis. Diversos trabalhos científicos já demonstraram as vantagens da utilização da simulação distribuída para obtenção de uma maior quantidade de resultados em um tempo mais curto (Yau & Pawlikowski, 2000; Wieland, 2001). O problema da simulação distribuída é a existência de mais um fator no já complexo processo citado no parágrafo anterior. Dessa forma, o usuário deve ser capaz de aplicar também os conceitos de computação paralela, envolvendo atividades como divisão do programa de simulação em processos, escalonamento e balanceamento dos processos, comunicação, etc.

Tendo em vista as dificuldades relacionadas nos parágrafos anteriores, este trabalho de doutorado propõe a especificação e implementa um protótipo de um ambiente automático para simulação distribuída.

O ambiente proposto, denominado ASDA (Ambiente de Simulação Distribuída Automático), possui as características apresentadas pelos ambientes de simulação automáticos (como por exemplo, o ASiA - descrito no capítulo anterior), isto é, uma interface amigável, geração e compilação automática do código de simulação, execução e análise dos resultados. Adicionalmente, o ASDA permite a utilização de simulação distribuída por usuários que não possuem profundos conhecimentos de computação paralela, dos métodos de sincronização para simulação distribuída ou dos métodos para análise dos resultados.

Um ponto forte do ASDA é sua flexibilidade. Uma opção oferecida é a geração de programas de simulação sequenciais e a utilização destes programas para execução paralela através da abordagem MRIP. Essa mesma abordagem pode ainda ser utilizada a partir de

programas de simulação seqüenciais já desenvolvidos pelo usuário. O usuário pode ainda optar pela abordagem SRIP, sendo que nesse caso os protocolos conservativos e otimistas podem ser considerados.

Este capítulo tem por objetivo apresentar uma descrição e a especificação formal do ASDA, e ao final, fazer uma comparação entre o ASDA e os ambientes automáticos descritos no capítulo anterior, seção 3.6.2.

4.2 ASDA

O ASDA é um ambiente de simulação voltado à avaliação de desempenho de sistemas computacionais. Como os modelos que melhor representam esses sistemas são os dinâmicos e contínuos no tempo, estocásticos e de estados discretos, estes são os tipos de modelos considerados no ambiente desenvolvido.

O público alvo para o qual o ASDA foi desenvolvido são alunos, professores e pesquisadores interessados na avaliação de desempenho de sistemas computacionais, principalmente os baseados na teoria de redes de filas.

O objetivo do ASDA é oferecer ao usuário um ambiente automático para o desenvolvimento de simulação, seguindo as características descritas nos ambientes automáticos. Os usuários de um ambiente desse tipo podem apresentar diferentes tipos de conhecimento, como por exemplo (Bruschi *et al.*, 2000):

- Usuários que apresentam um grande conhecimento em simulação, mas pouco conhecimento em computação paralela;
- Usuários que possuem conhecimento superficial tanto de simulação como de computação paralela;
- Usuários com 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.

O ASDA deve oferecer a flexibilidade necessária para auxiliar os diferentes perfis de usuários. Desta forma, alguns requisitos para o ASDA são (Bruschi *et al.*, 2000):

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 sequenciais 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.

O diagrama da estrutura modular proposto para o ASDA é apresentado na figura 4.1.

Para atingir os requisitos apresentados, o ASDA é composto por diversos módulos. Cada um destes módulos possui funções bem definidas com pode ser observado a seguir:

- **Módulo de Interface com o usuário:** responsável por oferecer ao usuário uma interface que possibilite 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. Este módulo é o maior responsável para a obtenção dos requisitos 1, 2 e 3;
- **Módulo Avaliador:** baseando-se nas variáveis do ambiente e no modelo a ser simulado, este módulo orienta o usuário na difícil tarefa de escolher a melhor abordagem para sua simulação. Neste módulo deve estar concentrado todo o conhecimento necessário para decidir ou para orientar o usuário sobre a melhor abordagem para a simulação, atendendo ao requisito 5;
- **Módulo Gerador:** baseando-se na especificação gráfica e na decisão do módulo avaliador, gera o código do programa de simulação, conforme requerido pelos requisitos 1 e 3;
- **Módulo Executor:** responsável por controlar a execução dos programas de simulação, atendendo ao requisito 1;
- **Módulo de Interface de Software:** responsável por fornecer uma interface entre os programas de simulação já desenvolvidos e o módulo replicador, atende ao requisito 4;
- **Módulo Replicador:** gerencia as replicações da simulação e garante a obtenção de resultados confiáveis através da abordagem MRIP, conforme requerido pelo requisito 6;
- **Módulo Escalonamento:** responsável pelo escalonamento dos processos gerados pelo módulo executor, atende ao requisito 7.

Figura 4.1. Diagrama Modular do ASDA

Dessa forma, qualquer que seja o usuário do ASDA, o módulo de interface com o usuário deverá ser utilizado. A partir desta interface, o usuário pode optar por: especificar um novo modelo de simulação; utilizar um programa de simulação já desenvolvido ou definir as variáveis de ambiente. Estas possibilidades oferecem a flexibilidade desejada pelos usuários mais experientes e oferecem todas as ferramentas requeridas pelos usuários menos experientes.

Se a opção escolhida for a utilização de um programa de simulação já existente, o módulo de interface de *software* é ativado, inserindo no código já existente as funções necessárias para replicar o programa, utilizando a abordagem MRIP.

Caso a opção escolhida seja o desenvolvimento de um novo modelo então o usuário

precisa especificar o modelo, tarefa esta realizada através do sub módulo de especificação gráfica. A consistência do modelo é verificada com o auxílio do nível 1 do módulo avaliador. O módulo avaliador é o módulo “inteligente” do sistema e auxilia também em outras decisões.

Baseando-se na especificação gráfica do modelo a ser simulado e nas informações oferecidas pelos níveis 2 e 3 do módulo avaliador, o módulo gerador gera o código do programa de simulação. O nível 2 do módulo avaliador auxilia na decisão entre as abordagens sequencial, SRIP, MRIP ou SRIP + MRIP. O nível 3 oferece os subsídios para a decisão entre os protocolos de sincronização SRIP. Três protocolos estão disponíveis para este caso: protocolo Time Warp, protocolo CMB e protocolo CMB/Time Warp. O CMB/Time Warp é uma adaptação proposta neste trabalho para a opção desenvolvida por Morselli (2000) e a ser implementada segundo especificado em Kawabata (2002) onde os protocolos de sincronização podem ser trocados em tempo de execução.

Após o código ter sido gerado, o programa é compilado e encaminhado para o módulo executor, o qual gerencia as execuções. Caso sejam escolhidas as abordagens MRIP ou SRIP + MRIP o módulo executor ativa o módulo replicador, o qual tem a função de gerenciar as replicações. O módulo executor é ainda encarregado de manter comunicação com o módulo escalonamento, que determina quais máquinas são mais adequadas para executar a simulação.

Com o objetivo de formalizar a especificação do ASDA, optou-se pela utilização da UML (*Unified Modeling Language*). É importante ressaltar que não foram utilizados todos os recursos disponíveis pela linguagem devido ao fato de este não ser o objetivo principal deste trabalho. O objetivo principal deste trabalho é a especificação do ASDA e a UML é utilizada como uma ferramenta para atingir esse objetivo.

No entanto, a modelagem é uma parte importante das atividades que levam a um bom *software*. Os modelos são simplificações da realidade e são construídos de forma que se possa compreender melhor o sistema que está sendo desenvolvido, explicitando a estrutura e o comportamento desejados.

Quando os sistemas são simples a modelagem formal² talvez nem seja necessária. Porém, nada impede que com o tempo, esses projetos simples tornem-se complexos e se não houver uma boa definição do sistema, este pode apresentar muitas falhas. Nos sistemas complexos a modelagem é essencial pois não é possível compreendê-lo em sua totalidade. Modelar o sistema ajuda a delimitar o problema (Booch *et al.*, 2000).

Desta forma, o início da modelagem do ASDA utilizando UML tem como objetivo definir melhor o projeto e facilitar sua continuação. Uma das vantagens de se utilizar UML é o fato de que a linguagem facilita a comunicação entre os vários desenvolvedores, além do fato de ser uma linguagem padrão de modelagem de sistemas.

Para o ASDA, foram utilizados 3 diagramas definidos pela UML:

² A maioria dos sistemas possuem alguma modelagem, apesar de ser informal.

- Diagrama de classes: é um diagrama estático do sistema composto por classes, interfaces, colaborações e relacionamentos de dependência, generalização e associação;
- Diagrama de casos de uso: é um diagrama comportamental do sistema composto por casos de uso, atores e relacionamentos de dependência, generalização e associação;
- Diagrama de atividades: é um diagrama comportamental do sistema composto por estados de atividades e estados de ação, transições e objetos.

Uma breve introdução a UML pode ser encontrada no Apêndice B.

As próximas seções discutem os objetivos de cada módulo e apresentam as especificações desses módulos.

4.3 Módulo de Interface com o usuário

O módulo de Interface com o usuário do ASDA tem por finalidade básica tornar fácil a interação do usuário com o ambiente. Para isso, deve seguir as características de desenvolvimento de uma boa interface com o usuário, as quais são descritas na subseção 4.3.1. Após a definição dessas características, as funções do módulo são definidas em 4.3.2 e em 4.3.3 é iniciada a formalização do ambiente utilizando a linguagem UML.

4.3.1 Interface com o Usuário

O termo “interface com o usuário” refere-se aos métodos e dispositivos que são utilizados na interação entre as máquinas e os seres humanos (definidos como usuários). A interface possui duas tarefas fundamentais: transmitir informação da máquina para o usuário e capturar informação do usuário para a máquina. Trata-se de um tópico importante pois, segundo Hix & Hartson (1993), para os usuários, a interface é o sistema.

Os dispositivos utilizados normalmente na interação com o usuário são: teclados e *mouses* (responsáveis por capturar as informações), monitor e impressora (responsáveis por transmitir as informações). Atualmente, dispositivos de interação mais modernos estão à disposição dos usuários, tais como: monitores *touchscreen*, câmaras de vídeo, microfones e alto-falantes.

Algumas características que fazem com que uma interface com o usuário seja definida como de boa qualidade são (Mayhew, 1992):

- Compatibilidade com o usuário: deve-se conhecer o usuário que irá manipular o software e adequar a interface ao perfil desse usuário;
- Compatibilidade com os produtos: as versões de um mesmo produto devem manter

interfaces similares de forma a não fazer o usuário perder muito tempo aprendendo a nova interface;

- Compatibilidade com o fluxo de tarefas: os sistemas devem ser organizados de forma a facilitar a transição entre as diferentes tarefas disponíveis;
- Consistência: as operações semelhantes disponíveis em diferentes aplicações no sistema devem possuir interfaces similares ou até mesmo idênticas;
- Familiaridade: conceitos, terminologias e disposições espaciais de objetos que já sejam familiares ao usuários devem ser inseridos na interface;
- Simplicidade: disponibilizar as tarefas simples de uma maneira simples, sem oferecer muitas opções que podem nem ser utilizadas; e as tarefas complexas de uma maneira possível de ser realizada;
- Manipulação direta: característica em que os usuários realizam as ações diretamente em objetos visíveis, sem ser através de comandos ou parâmetros;
- WYSIWYG (*What You See Is What You Get*): grau de correspondência entre a informação que o usuário visualiza na tela e o que é armazenado ou impresso. Quanto maior esse grau, melhor a interface;
- Flexibilidade: permite que o usuário tenha um controle maior de acordo com suas preferências, permitindo possibilidades de escolha;
- *Feedback*: o sistema deve sempre responder imediatamente a uma requisição do usuário. Se for uma operação em que o tempo de resposta é longo, o sistema deve informar o progresso da requisição;
- Robustez: o sistema deve ser tolerante a falhas dos usuários, sendo que as quebras do sistema devem ser minimizadas. Medidas de recuperação devem ser fáceis de se entender e executar;
- Proteção: deve-se proteger o sistema de prováveis erros comuns, tais como apagar todos os arquivos de um diretório, e prover mecanismos de recuperação desses erros;
- Facilidade de aprendizado e facilidade de uso: o sistema deve ser de fácil aprendizado para os novos usuários e de fácil utilização para os usuários mais experientes.

4.3.2 Descrição das atividades do Módulo de Interface

O módulo de interface do ASDA deve atingir os seguintes objetivos:

- Oferecer ao usuário uma interface amigável;

- Identificar e gerenciar os usuários do sistema;
- Receber os dados definidos pelo usuário e armazená-los em bases de dados;
- Disponibilizar os dados armazenados para os usuários e para os outros módulos do sistema;
- Permitir a especificação gráfica do modelo a ser simulado.

Com o intuito de atingir o primeiro objetivo citado, a interface com o usuário do ASDA deve utilizar os conceitos de janelas, menus, caixas de diálogo, barras de ferramentas e barras de rolagem, os quais são amplamente conhecidos devido aos ambientes multitarefas, tais como Microsoft Windows, Apple Macintosh e os ambientes gráficos do Linux (KDE e Gnome).

Diversas telas devem estar disponíveis para a utilização do sistema. A tela inicial deve permitir ao usuário selecionar o tipo de utilização que pretende fazer do ASDA, identificando-se. A cada solicitação novas telas devem ser ativadas. Uma melhor definição destas telas é apresentada no capítulo 5.

A segunda tarefa do módulo de interface é identificar e gerenciar os usuários que utilizam o sistema. Existem dois níveis de usuários:

- Super-usuário: responsável por determinar os valores padrões das variáveis do ambiente e também por cadastrar novos usuários. A esse tipo de usuário não é permitido utilizar o ambiente para realizar simulações;
- Usuário de simulação: usuário que irá realizar simulações.

Os objetivos 3 e 4 listados na seção 4.2 relacionam-se com o armazenamento e disponibilização dos dados relacionados ao sistema e ao usuário, isto é, sendo responsável pela interação do ambiente com o usuário, o módulo de Interface com o usuário do ASDA deve obter do usuário e/ou dos outros módulos as informações que serão armazenadas na base de dados, além de fornecer para o usuário e/ou outros módulos as informações já armazenadas.

Os dados armazenados no ASDA podem ser subdivididos em dados do usuário e dados do sistema. Com a finalidade de estabelecer perfis de usuários, a base de dados referente 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 faça *login* 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;

Conforme o usuário vai definindo 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 figura 4.2 ilustra graficamente o exemplo de um arquivo de usuário. Nesse exemplo temos que o usuário *João*, que possui a senha *j5ng4*, prefere: modelar seus sistemas utilizando redes de filas; utilizar a extensão funcional SMPL; utilizar a abordagem MRIP; deixar que o programa defina o tamanho do warm-up e o tamanho do lote (indicamos pelo número -1). Na lista de modelos, o primeiro modelo é identificado por *modelo1* sendo que a especificação gráfica desse modelo está armazenada no arquivo *mml.esp* e o programa da simulação no arquivo *mml.c*. A abordagem escolhida para realizar a simulação foi a SRIP (protocolo CMB) com a extensão funcional ParSMPL. A simulação foi executada numa rede de computadores pessoais e os resultados armazenados no arquivo *mml.out* e estes foram gerados com nível de confiança 95% e precisão 5%. O modelo *modelo4* utiliza as definições padrões do usuário e para o *modelo10* foi realizada somente a especificação gráfica.

Figura 4.2. Exemplo de um registro do arquivo de usuários

O arquivo de especificação gráfica deve armazenar as informações referentes ao modelo definido pelo usuário. O modelo é definido através do sub-módulo *Especificação Gráfica* que tem por finalidade permitir que o usuário utilize os recursos disponíveis para representar seu sistema através de um modelo.

A especificação gráfica é composta basicamente por uma região onde o modelo é descrito, denominada janela para especificação, e por barras de ferramentas que contêm os recursos disponíveis para a especificação do modelo. Estes recursos dependem da abordagem escolhida. Como descrito na seção 2.3.5, *Redes de Filas estendida* e *Queueing Statecharts* foram escolhidas para serem utilizadas na representação dos modelos no ASDA. Esta escolha se deve principalmente à facilidade que estas abordagens possuem para representar modelos de sistemas computacionais, os quais apresentam a característica de possuir servidores e filas.

No caso da abordagem *Redes de Filas estendida*, a barra de ferramentas deve conter os objetos para a definição de centros de serviço com uma fila e um servidor, várias filas e um servidor, uma fila e vários servidores, várias filas e vários servidores e o servidor infinito. Além dessas opções de centros de serviço, o usuário pode ainda utilizar o servidor sem fila e os centros passivos (necessários para a implementação da posse múltipla).

Já na utilização do *Queueing Statecharts*, os recursos disponíveis são os *templates* descritos no capítulo 2, seção 2.3.4, com a possibilidade de expandir a definição básica de cada *template*. No caso da expansão, torna-se disponível a definição de novos estados e

transições, e mudanças nas regras de transição.

Conforme o usuário vai definindo seu modelo é necessária a verificação da consistência do modelo de modo a evitar problemas na geração do código. Essa tarefa é realizada pelo módulo Avaliador, nível 1, o qual contém as regras necessárias para essa verificação. As regras são descritas na subseção 4.3.4, quando é detalhado o módulo Avaliador.

Para facilitar a análise do modelo descrito pelo usuário na janela de especificação para uma posterior decomposição em processos lógicos, este é armazenado na forma de um grafo. Cada centro de serviço definido pelo usuário recebe um número e é representado por um nó do grafo; as ligações entre os centros de serviços são representadas pelos arcs. Quando existem probabilidades, estas são representadas como os pesos dos arcs. Um exemplo de uma transcrição de um modelo de redes de filas para grafos pode ser observado na figura 4.3.

Figura 4.3. Representação de um modelo de Rede de Filas através de Grafos

Existem cinco tipos de nós: o nó centro de serviço, o nó entrada, o nó saída, o nó servidor sem fila e o nó posse múltipla. Uma associação entre a representação gráfica dos recursos em Redes de Filas e *Queueing Statechart* e os nós do grafo pode ser observada na tabela 4.1. É importante ressaltar que independente da abordagem a ser utilizada pelo usuário (Redes de Filas estendida ou *Queueing Statechart*), as informações a serem armazenadas pelo módulo de Especificação Gráfica são as mesmas.

Definido o modelo, passa-se à fase de parametrização, a qual consiste na definição dos parâmetros de cada nó e do modelo em geral.

Em relação aos nós, para cada tipo de nó são armazenadas diferentes informações, como pode ser observado na tabela 4.2. O nó posse múltipla deve sempre aparecer aos pares, um indicando a alocação do recurso passivo ao cliente e o outro o indicando a liberação do recurso.

Tabela 4.1. Associação entre a representação gráfica utilizando Redes de Filas, *Queueing Statecharts* e o grafo a ser armazenado

Recurso	Redes de Filas	<i>Queueing Statecharts</i>	Grafo
Fonte	Recurso		NE
Centro de serviço			CS
Destino	Recurso		NS
Posse múltipla	e	----	NPM
Servidor sem fila		----	NSSF

Em relação ao modelo, é necessário armazenar:

- 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).

Tabela 4.2. Informações a serem armazenadas em cada nó do modelo

Nó	Descrição	
Centro de Serviço	<i>Identificador:</i>	cada centro de serviço deve ser identificado por um número, que funcionará como chave primária e será definido pelo sistema
	<i>Nome do centro de serviço:</i>	nome do centro de serviço
	<i>Número de filas:</i>	número de filas que compõem o centro de serviço
	<i>Número de servidores:</i>	número de servidores que compõem o centro de serviço
	<i>Distribuição de serviço:</i>	distribuição de probabilidade do tempo para que um servidor execute um serviço;
	<i>Parâmetro a ser analisado:</i>	determina qual(quais) parâmetro(s) do centro de serviço vai(vão) ser analisado(s). Esses parâmetros podem ser: utilização do servidor, comprimento médio da fila de espera, tamanho máximo da fila, tamanho máximo de espera na fila, tempo médio de espera no sistema, entre outros.
Saída	<i>Identificador:</i>	identificador do nó saída
Entrada	<i>Identificador:</i>	identificador do nó entrada
	<i>Distribuição de chegada:</i>	distribuição de probabilidade do tempo entre as chegadas dos clientes
Posse Múltipla	<i>Identificador:</i>	identificador do nó posse múltipla
	<i>Nome do recurso passivo:</i>	nome do recurso passivo que irá ser representado
	<i>Tipo do nó posse múltipla:</i>	<i>alocação</i> caso o nó esteja representando uma alocação e <i>liberação</i> caso o nó esteja representando uma liberação
Servidor sem fila	<i>Identificador:</i>	identificador do nó servidor sem fila
	<i>Nome do servidor:</i>	nome do servidor sem fila
	<i>Política de reescalonamento:</i>	política a ser seguida para uma nova tentativa de utilização do recurso caso ele esteja ocupado
	<i>Distribuição de serviço:</i>	distribuição de probabilidade do tempo para que o servidor execute um serviço;

A base de dados referente às informações do usuário é definida como a que contém as informações de cada usuário, incluindo os modelos definidos, e a base de dados referente às informações do sistema é definida como sendo a que contém informações referentes à configuração do ambiente e também informações referentes aos módulos Avaliador, Executor, Replicador e Escalonamento.

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 (CMB, Time Warp, CMB/Time Warp) 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;

4.3.3 Especificação do Módulo de Interface em UML

O Diagrama de Casos de Uso do ASDA, que formaliza o comportamento descrito na seção 4.3.2, é apresentado na figura 4.4. Nesse diagrama pode-se observar os dois tipos de usuários do sistema e quais casos de uso estão disponíveis para cada usuário.

Figura 4.4. Diagrama de Casos de Uso do ASDA

Os casos de uso relacionados com o Módulo de Interface são detalhados nesta seção.

Neste caso, enquadram-se:

- Validar usuários;
- Adicionar novos usuários;
- Definir variáveis do ambiente (extends: Definir variáveis módulo Avaliador, Definir variáveis módulo Executor, Definir variáveis módulo Replicador e Definir variáveis módulo Escalonamento);
- Desenvolver uma nova simulação.

Os demais casos de uso serão detalhados nas próximas seções, quando os módulos aos quais estão relacionados forem descritos.

Caso de Uso: Validar Usuário

Descrição: Permite que o usuário inicie a utilização do ambiente, identificando seu nome e senha.

Fluxo Principal de Eventos:

Ação do Ator	Resposta do Sistema
1. O usuário inicia a execução do ambiente.	
	2. Fornece a tela de identificação
3. O usuário se identifica, fornecendo o nome e a senha.	
	4. Procura pelo nome do usuário na tabela de usuários.
	5. Decodifica a senha armazenada e compara com a fornecida pelo usuário.
6. O usuário inicia a utilização do ambiente.	

Fluxo Alternativo de Eventos:

Linha 1: o usuário cancela a operação

Linha 2: o usuário fornece o nome errado

Linha 2: o usuário fornece a senha errada

Linha 3: o sistema não localiza o usuário

Linha 4: o sistema não consegue decodificar a senha

Pós-condição: O usuário está habilitado a utilizar o ambiente

Caso de Uso: Adicionar novos usuário

Descrição: Permite ao usuário adicionar novos usuários ao sistema e alterar as informações dos existentes.

Pré-condição: Ter permissão para ser SuperUsuario; caso opte por alterar os dados de algum usuário, este deve estar cadastrado.

Fluxo Principal de Eventos:

Ação do Ator	Resposta do Sistema
1. O usuário seleciona entre: <ul style="list-style-type: none"> a. Adicionar um novo usuário b. Alterar as informações de um usuário já cadastrado 	
	2. Fornece a janela onde serão fornecidos os dados
3. Fornece ou altera os dados	
	4. Efetua as alterações

Fluxo Alternativo de Eventos:

Linha 1: O usuário cancela a operação

Linha 4: O sistema não consegue armazenar as informações

Pós-condição: Ter um novo usuário cadastrado ou com seus dados alterados

Caso de Uso: Definir as variáveis do ambiente

Descrição: Permite ao usuário a definição das variáveis do ambiente

Pré-condição: Ter permissão para ser SuperUsuario.

Fluxo Principal de Eventos:

Ação do Ator	Resposta do Sistema
1. O usuário seleciona a opção de definir as variáveis do ambiente	
	2. Fornece a tela com as variáveis que podem ser alteradas
3. Altera os valores das variáveis	
	4. Valida as informações fornecidas
	5. Armazena

Fluxo Alternativo de Eventos:

Linha 3: O usuário cancela a operação

Linha 4: As informações não estão coerentes

Extend:

Linha 3: Definir variáveis módulo avaliador

Linha 3: Definir variáveis módulo executor

Linha 3: Definir variáveis módulo replicador

Linha 3: Definir variáveis módulo escalonamento

Pós-condição: As variáveis do ambiente estão alteradas.

Caso de Uso: Definir as variáveis módulo Avaliador

Descrição: Permite ao usuário definir as variáveis do módulo avaliador, tais como as condições pelas quais as abordagens SRIP ou MRIP são escolhidas e também a escolha entre os protocolos Time Warp ou CMB.

Pré-condição: Ter permissão para ser SuperUsuario.

Fluxo Principal de Eventos:

Ação do Ator	Resposta do Sistema
1. O usuário seleciona alterar as variáveis do módulo Avaliador	
	2. Fornece a tela com as variáveis que podem ser alteradas
3. Altera os valores das variáveis	
	4. Valida as informações fornecidas
	5. Armazena

Fluxo Alternativo de Eventos:

Linha 3: O usuário cancela a operação

Linha 4: As informações não estão coerentes

Pós-condição: As variáveis do módulo Avaliador estão alteradas.

Caso de Uso: Definir as variáveis do módulo Executor

Descrição: Permite ao usuário definir as variáveis do módulo executor, tais como a arquitetura e o conjunto de máquinas sobre qual a simulação será executada.

Pré-condição: Ter permissão para ser SuperUsuario.

Fluxo Principal de Eventos:

Ação do Ator	Resposta do Sistema
1. O usuário seleciona alterar as variáveis do módulo Executor	
	2. Fornece a tela com as variáveis que podem ser alteradas
3. Altera os valores das variáveis	
	4. Valida as informações fornecidas 5. Armazena

Fluxo Alternativo de Eventos:

Linha 3: O usuário cancela a operação

Linha 4: As informações não estão coerentes

Pós-condição: As variáveis do módulo Executor estão alteradas.

Caso de Uso: Definir as variáveis do módulo Replicador

Descrição: Permite ao usuário definir as variáveis do módulo replicador, tais como o tamanho do período transiente, o método de análise de saída e o tamanho do batch.

Pré-condição: Ter permissão para ser SuperUsuario.

Fluxo Principal de Eventos:

Ação do Ator	Resposta do Sistema
1. O usuário seleciona alterar as variáveis do módulo Replicador	
	2. Fornece a tela com as variáveis que podem ser alteradas
3. Altera os valores das variáveis	
	4. Valida as informações fornecidas 5. Armazena

Fluxo Alternativo de Eventos:

Linha 3: O usuário cancela a operação

Linha 4: As informações não estão coerentes

Pós-condição: As variáveis do módulo Replicador estão alteradas.

Caso de Uso: Definir as variáveis do módulo Escalonamento

Descrição: Permite ao usuário definir as variáveis do módulo escalonamento, tais como a política utilizada para o escalonamento e a aplicação que será executada em cada máquina pertencente à máquina virtual para obtenção da potência computacional.

Pré-condição: Ter permissão para ser SuperUsuario.

Fluxo Principal de Eventos:

Ação do Ator	Resposta do Sistema
1. O usuário seleciona alterar as variáveis do módulo Escalonamento	
	2. Fornece a tela com as variáveis que podem ser alteradas
3. Altera os valores das variáveis	
	4. Valida as informações fornecidas
	5. Armazena

Fluxo Alternativo de Eventos:

Linha 3: O usuário cancela a operação

Linha 4: As informações não estão coerentes

Pós-condição: As variáveis do módulo Escalonamento estão alteradas.

Caso de Uso: Desenvolver uma nova simulação

Descrição: Permite ao usuário definir uma nova simulação utilizando as abordagens MRIP ou SRIP contando com o auxílio do ambiente ou não para tomar essa decisão.

Pré-condição: Ter permissão para ser UsuarioSimulacao.

Fluxo Principal de Eventos:

Ação do Ator	Resposta do Sistema
1. O usuário seleciona desenvolver uma nova simulação	
	2. Fornece a tela de especificação gráfica para o desenvolvimento e parametrização do modelo
3. Especifica e parametriza o modelo	
	4. Valida as informações fornecidas 5. Pergunta ao usuário se ele deseja escolher a abordagem para gerar o programa de simulação ou se precisa de auxílio do ambiente
6. Escolhe entre obter auxílio ou não	
	7. Fornece o auxílio caso este foi requisitado

Fluxo Alternativo de Eventos:

Linha 3: O usuário cancela a operação

Linha 4: As informações não estão coerentes

Linha 5: O usuário não deseja gerar o programa de simulação

Linha 6: O usuário cancela a operação

Extend:

Linha 7: Utilizar o auxílio do ambiente

Pós-condição: Definição de um novo modelo a ser simulado.

4.4 Módulo Avaliador

O módulo Avaliador tem por finalidade auxiliar o usuário verificando as informações fornecidas e também auxiliando nas várias decisões que devem ser tomadas no sistema. Desta forma, o módulo Avaliador foi dividido em três níveis, sendo:

- Nível 1: verifica se o modelo do usuário está sendo especificado corretamente;
- Nível 2: ajuda o usuário a escolher entre simulação sequencial ou distribuída SRIP ou MRIP;
- Nível 3: ajuda o usuário a escolher entre simulação distribuída SRIP utilizando o protocolo CMB, protocolo Time Warp ou considerando a troca de protocolos em

tempo de execução (CMB $\rightarrow$ Time Warp). Este nível foi ainda subdividido em dois sub-níveis os quais são encarregados de tratar a divisão em processos lógicos nos protocolos CMB, Time Warp e CMB $\rightarrow$ Time Warp.

A avaliação a ser realizada pelo módulo Avaliador nos níveis 2 e 3 baseia-se na plataforma onde será executada a simulação, no modelo e na sua parametrização, e nos parâmetros da simulação.

É importante observar que os níveis 2 e 3 deste módulo não fazem uso de métricas obtidas em tempo de execução. As análises apresentadas nas sub seções 4.4.2 e 4.4.3 são baseadas em resultados já apresentados na literatura e, como será descrito na sub seção 4.4.4, utilizará algumas técnicas de Inteligência Artificial para chegar às conclusões. Assim, os níveis 2 e 3 do módulo Avaliador fazem uma análise estática do modelo, do ambiente e da plataforma a ser utilizada para decidir sobre a abordagem mais adequada.

As próximas sub seções descrevem com maiores detalhes os níveis 1, 2 e 3 do módulo Avaliador e também o caso de uso da figura 4.4 referente a este módulo.

4.4.1 Nível 1

O nível 1 do módulo Avaliador atua em conjunto com o módulo de Especificação Gráfica. A verificação da consistência do modelo fornecido pelo usuário é realizada conforme o usuário vai definindo o modelo. Um dos motivos pelos quais optou-se por armazenar o modelo em forma de grafo foi a facilidade que essa estrutura fornece (através dos algoritmos já desenvolvidos) na fase de verificação do modelo, além de facilitar também a geração do código.

Considerando a representação através de Redes de Filas estendidas ou *Queueing Statecharts*, as seguintes regras para definição do modelo devem ser seguidas:

- Todo centro de serviço deverá ter pelo menos um caminho por onde os clientes entram e outro por onde os clientes saem. Analisando o grafo do modelo, os únicos nós que podem ter somente uma conexão são os nós entrada e saída;
- Todos os centros de serviço devem estar conectados entre si de alguma forma, ou seja, não poderá haver dois ou mais conjuntos de centros de serviço. Analisando o grafo, deve-se verificar se não existem nós sem nenhuma conexão;
- Deve haver sempre um par de elementos que indicam a posse múltipla para recursos passivos e estes devem estar conectados.

Com relação à parametrização, este módulo tem por finalidade verificar se todos os parâmetros foram fornecidos e também alertar o usuário em situações onde os valores dos parâmetros podem levar a resultados incoerentes na simulação, como por exemplo, a definição de um tempo de serviço muito maior do que o tempo entre as chegadas.

Essas regras definem como o usuário deve especificar o modelo de forma que ele possa ser codificado sem gerar erros.

O ASiA (Bruschi, 1997; Kawabata, 1999) possui uma interface para especificação de modelos utilizando redes de filas que verifica a consistência dos modelos conforme estes vão sendo definidos. As regras descritas acima foram definidas a partir da verificação existente no ASiA.

4.4.2 Nível 2

O nível 2 do módulo Avaliador auxilia o módulo Gerador. Caso o usuário solicite, o nível 2 do módulo Avaliador tem por finalidade avaliar o modelo e a arquitetura sobre a qual a simulação será executada e sugerir a utilização das abordagens SRIP ou MRIP. A informação gerada por este nível deve alimentar o módulo Gerador.

A grande diferença entre as abordagens SRIP e MRIP está na granulação dos processos paralelos gerados e consequentemente no número de processos.

No caso da simulação MRIP, a simulação do modelo é feita de maneira sequencial e as replicações do programa de simulação são executadas paralelamente. Esta abordagem permite uma granulação normalmente mais grossa e um pequeno número de comunicações. Isto é, como todo programa de simulação é executado em um único programa, apenas os resultados da simulação devem ser enviados para o processo mestre.

No caso da simulação SRIP, a simulação do modelo é particionada em diversos processos, levando à necessidade de protocolos de sincronização para evitar os erros de “causa e efeito”. Essa necessidade de sincronização leva à necessidade de um número de comunicações bastante elevado. Dependendo da granulação obtida na partição dos processos e da plataforma onde está sendo executada a simulação, as comunicações necessárias podem prejudicar o desempenho da simulação distribuída.

É importante observar que apesar da abordagem SRIP apresentar uma sobrecarga muito maior que a abordagem MRIP, o SRIP apresenta uma flexibilidade muito maior. Isto é, a utilização da abordagem SRIP permite explorar totalmente o paralelismo da simulação enquanto que a abordagem MRIP não permite explorar qualquer paralelismo.

Baseando-se no que foi discutido nos últimos parágrafos pode-se concluir que a opção por MRIP ou SRIP depende da granulação obtida pelos processos e da plataforma a ser utilizada.

Considerando-se que os processos SRIP possuem uma granulação mais fina que os processos MRIP, pode-se, em uma primeira análise, considerar a distribuição proposta na figura 4.5. Neste caso, as plataformas do tipo 2 oferecem comunicação rápida e processamento não muito eficiente, sendo mais adequadas para a abordagem SRIP. As plataformas do tipo 3 possuem comunicação não tão eficiente mas processamento rápido,

levando a adequação para a abordagem MRIP. As plataformas do tipo 1, a princípio, poderiam ser utilizadas tanto para SRIP como para MRIP.

Figura 4.5. Escolha entre as abordagens SRIP e MRIP segundo a plataforma (processamento x comunicação)

As plataformas do tipo 3 podem ser exemplificadas com a arquitetura das redes Ethernet, onde pode-se ter uma configuração com diversas máquinas velozes porém conectados por uma rede pouco eficiente. Já as plataformas do tipo 2 podem ser exemplificadas por máquinas paralelas que possuam uma rede de comunicação muito eficiente mas processadores não tão velozes.

Este processo deve ser refinado considerando outros fatores, tais como os citados por Heidelberg (1986):

- tamanho do período transiente;
- número e a capacidade dos processadores;

e também por outros fatores tais como:

- grau de paralelismo;
- tamanho e o detalhamento do modelo;
- a precisão requerida;
- os parâmetros do modelo.

Estes fatores são de grande importância, pois podem alterar as classificações apresentadas na figura 4.5 e são essenciais para as plataformas do tipo 1, que a princípio são tão adequadas para SRIP quanto para MRIP.

De uma forma resumida, Heidelberg em seu modelo estatístico (Heidelberg, 1986)

conclui que se o período transiente é forte e o tamanho da simulação é pequeno a melhor opção é a abordagem SRIP. Já se o período transiente é fraco e o tamanho da simulação é longo a melhor abordagem a ser utilizada é a MRIP (seção 3.4). A figura 4.6 ilustra essas conclusões sendo que na diagonal pode-se optar por qualquer uma das abordagens, levando-se em consideração outro fatores.

Figura 4.6. Escolha entre as abordagens SRIP e MRIP segundo a duração da simulação e a duração do *warm-up*

O grau de paralelismo do modelo pode ser medido através da análise do grafo gerado a partir do modelo. Se o grau de paralelismo for alto, a abordagem SRIP é beneficiada pois ela explora bem o paralelismo implícito nos modelos. Já para a abordagem MRIP, o grau de paralelismo não influencia seu desempenho. Para modelos em que o grau de paralelismo for muito pequeno, com a presença de muita realimentação, a utilização da abordagem SRIP pode se tornar inviável. Neste caso, a sobrecarga imposta pelo protocolo de sincronização leva a tempos de simulação muito elevados.

Considerando o tamanho do modelo pode-se observar que conforme este aumenta, aumenta o número de centros de serviço. Considerando um número fixo e não muito grande de processadores, o aumento do número de centros de serviço implica em uma melhor adequação da abordagem SRIP, como pode ser observado na figura 4.7. Isto se deve ao fato de que, considerando a execução de um programa seqüencial, este se torna muito pesado para ser executado nos processadores em um único programa, tornando a granulação da aplicação mais grossa e prejudicando a abordagem MRIP.

É importante salientar que nem sempre um modelo simples leva a uma simulação de curta duração. Se resultados com alta precisão são necessários deve-se ter um maior número de observações. Assim a precisão dos resultados requerida pelo usuário também influencia na escolha entre as abordagens SRIP e MRIP pois quanto maior o grau de precisão, maior é o número de observações que devem ser coletadas e consequentemente, maior é a duração da

simulação. Neste caso, deve-se seguir as conclusões citadas nos parágrafos anteriores e que podem ser visualizadas na figura 4.6.

Figura 4.7. Escolha entre as abordagens SRIP e MRIP segundo o tamanho do modelo e o número de processadores

A relação existente entre o tamanho do modelo, a precisão requerida e a duração da simulação pode ser observada na figura 4.8. O gráfico pode ser analisado segundo as quatro retas desenhadas. A reta preta sólida indica que o tamanho da simulação, ou seja, o tempo de execução da simulação, aumenta proporcionalmente com o aumento do tamanho do modelo e da precisão. A reta preta tracejada indica que, mantendo em médio o tamanho do modelo, o tamanho da simulação também aumenta proporcionalmente com o aumento da precisão. Da mesma forma, a reta preta pontilhada indica que, mantendo a precisão, o tamanho da simulação aumenta proporcionalmente com o aumento do tamanho do modelo. Já a reta preta com traços e pontos indica um tamanho da simulação normal para relações inversas (tamanho do modelo grande e precisão pequena ou tamanho do modelo pequeno e precisão alta).

4.4.3 Nível 3

O nível 3 do módulo Avaliador, assim como o nível 2, auxiliará o módulo Gerador, porém com uma função um pouco diferente. O nível 3 será acionado somente se o usuário ou o nível 2 optarem pela utilização de simulação SRIP e auxiliará na decisão entre a utilização dos protocolos CMB, Time Warp ou CMB/Time Warp.

Figura 4.8. Relação entre o tamanho do modelo, a precisão requerida e o tamanho da simulação

Como descrito na seção 3.2.4, vários fatores influenciam o desempenho dos protocolos conservativos e otimistas. Tentar levar em consideração todos os fatores seria uma tarefa difícil, pois nem sempre todos os dados estão disponíveis para que possam ser utilizados e também porque é complicado definir todas as possibilidades. Além disso, existem alguns protocolos que são projetados especificamente para uma determinada aplicação, o que torna realmente inviável a definição de regras gerais.

Em linhas gerais, no entanto, algumas regras podem ser definidas de modo a associar determinadas características do modelo e da plataforma onde a simulação será executada a um determinado protocolo, alcançando um melhor desempenho da simulação distribuída. Nesta seção será considerado que os protocolos conservativo CMB e o otimista Time Warp são utilizados em suas versões originais.

Uma primeira tarefa desse nível do módulo avaliador é verificar qual a melhor forma de particionar o modelo em processos lógicos. Isto se deve ao fato de que, independente do protocolo de sincronização a ser utilizado, a forma de particionar o modelo em processos lógicos deve seguir algumas regras gerais. Deve-se também ao fato de que as variáveis analisadas como critérios de decisão pela segunda tarefa do nível 3 do módulo avaliador (selecionar o protocolo) dependem de forma considerável de como o modelo foi particionado.

Com o objetivo de realizar uma partição eficiente, várias pesquisas utilizam ferramentas analíticas, como a análise do caminho crítico (*critical path analysis*) (Berry & Jefferson, 1985; Lin, 1992; Overeinder & Sloot, 1995), e outras utilizam ferramentas experimentais (Swope & Fujimoto, 1987; Jha & Bagrodia, 1996). Algumas dessas ferramentas necessitam que a simulação seja executada e que seja gerado um *trace* para que possa identificar a

dependência dos eventos.

Como já enfatizado no início dessa seção, as análises apresentadas aqui são compilações de resultados já apresentados na literatura. Desse modo, algumas regras para a partição do modelo em processos lógicos podem ser obtidas levando em consideração o grafo do modelo (nível de paralelismo), a parametrização e a plataforma onde a simulação será executada (Wagner & Lazowska, 1989; Ulson, 1999; Dota, 2001; Spolon, 2001). Analisando-se os resultados e as regras apresentadas nesses trabalhos chegou-se a estas diretrizes básicas para a divisão do modelo em processos lógicos:

- Deve-se evitar o máximo possível que as realimentações (*feedback*) sejam colocadas em processos lógicos distintos, pois isso prejudica tanto o protocolo CMB (aumento do número de mensagens nulas) como o protocolo Time Warp (aumento dos erros de causa e efeito e consequentemente do número de *rollbacks*). Caso isso não seja possível, deve-se evitar que as realimentações que possuem grandes probabilidades de retorno estejam em processos lógicos distintos. Um exemplo dessa regra pode ser observado nas figuras 4.9a e 4.9b, onde a partição correta está em 4.9a.

Figura 4.9. Partição do modelo em processos lógicos analisando a realimentação

- A granulação do modelo deve ser proporcional à velocidade de comunicação da plataforma, ou seja, caso a rede de comunicação possua uma velocidade de comunicação alta então é permitido um número maior de processos lógicos e caso contrário, é melhor utilizar um número menor de processos lógicos. Isso pode ser observado quando se utiliza plataformas distribuídas, como por exemplo NOWs

(*Network of Workstations*) e *clusters*, onde a velocidade de comunicação é mais baixa e arquiteturas paralelas, como por exemplo IBM SP2, onde a velocidade de comunicação é mais alta.

- Deve-se evitar colocar em um mesmo processo lógico recursos que não se comunicam diretamente. Esse fato faz com que aumente consideravelmente o número de mensagens que vão circular no sistema. Um exemplo dessa característica pode ser observada nas figuras 4.10a e 4.10b, ilustrada em (Ulson, 1999). Neste caso, em 4.10b não existe comunicação direta nos recursos que estão em PL0 nem em PL1.

Figura 4.10. Partição do modelo em processos lógicos analisando

Definidas as partições, a segunda tarefa do nível 3 do módulo avaliador é decidir pela utilização dos protocolos CMB, Time Warp ou CMB/Time Warp.

De uma maneira geral, duas características influenciam fortemente a escolha entre os protocolos CMB e Time Warp. São elas: *lookahead* e número de *rollbacks*. O *lookahead* é muito importante em protocolos que utilizam o conceito de evitar o *deadlock* e, portanto, influencia fortemente o protocolo CMB. Por sua vez, o número de *rollbacks* é muito importante nos protocolos que consideram que os erros de causa e efeito não acontecem e,

portanto, influencia fortemente o protocolo Time Warp.

Graficamente, a escolha entre os protocolos CMB, Time Warp ou CMB/Time Warp pode ser observada na figura 4.11. A escolha pelo protocolo Time Warp é baseada no baixo número de *rollbacks* enquanto a escolha pelo protocolo CMB é baseada no alto grau de *lookahead*. Quando essas duas características são atingidas ao mesmo tempo, pode-se optar pela utilização do CMB ou do Time Warp. Já nos outros 4 casos aconselha-se a utilização do protocolo CMB/Time Warp.

Figura 4.11. Escolha entre os protocolos CMB, Time Warp e CMB $\Rightarrow$ Time Warp

O protocolo CMB/Time Warp é uma adaptação da abordagem proposta por Morselli (2000) em sua tese de doutorado. Nesse trabalho, Morselli propõe um novo mecanismo que possibilita a escolha entre os protocolos de sincronização para simulação distribuída, CMB e Time Warp, durante a execução de uma simulação distribuída baseada em redes de filas. Esta abordagem é interessante pois, como pode ser observado na figura 4.11, existem situações onde não existe a melhor opção na escolha do protocolo de sincronização.

Desta forma, torna-se atrativa a opção de iniciar a simulação com um protocolo e permitir que este seja trocado por outro protocolo automaticamente e em tempo de execução. A troca de protocolos deve considerar duas questões principais: como realizar a troca, uma vez que cada protocolo apresenta suas próprias características, e quando a troca deve ser considerada. Baseando-se nessas duas questões, o mecanismo para troca de protocolos proposto no trabalho de Morselli (2000) pode ser implementado através da execução dos processos lógicos da simulação concorrentemente com os processos: observador (responsável pela coleta de dados relativos ao desempenho da simulação), gerenciador (detecta a necessidade da troca de protocolos) e o conversor (responsável pela conversão entre os protocolos). O trabalho de Kawabata (2002) estuda a utilização de agentes móveis e lógica fuzzy para a implementação dos processos definidos por Morselli.

Para realizar a troca do protocolo Time Warp para CMB em tempo de execução é

necessário antes da troca colocar a simulação em uma situação estável e o custo para realizar esta operação é muito alto pois deve-se realizar um *rollback* nos processos mais adiantados. Por outro lado, a troca de CMB para Time Warp apresenta um menor custo e por este motivo optou-se pela utilização no ASDA. No ASDA, esta possibilidade será utilizada quando o ganho de desempenho com a utilização do CMB ou do Time Warp não estiver claro. Neste caso, deve-se optar pelo CMB com possibilidade de troca para o Time Warp. Se for necessário realizar a troca, o tempo despendido com a troca será menor do que se fosse escolhido Time Warp com eventual troca para CMB.

A grande dificuldade na escolha entre os protocolos de sincronização é que o cálculo das variáveis *rollback* e *lookahead* antes da execução da simulação não é simples. Um exemplo de como o *lookahead* pode ser calculado é apresentado em (Wagner & Lazowska, 1989). Nos modelos baseados em redes de filas com disciplina de fila FCFS (*First Come, First Served*), o cálculo do *lookahead* pode ser feito observando o valor das taxas de serviço dos servidores e da taxa de chegada. Caso um cliente esteja sendo atendido em um servidor então não haverá nenhuma partida até que este cliente termine o seu serviço. Desse modo, o valor restante do tempo de serviço pode ser utilizado como o *lookahead* para este servidor. Caso não exista nenhum cliente sendo atendido, o *lookahead* é igual a um valor a , que representa o valor mínimo da distribuição de serviço, o qual pode ser zero. Uma outra forma de calcular o *lookahead* é através da utilização de uma lista futura, na qual, caso não haja nenhum cliente sendo atendido, pode-se predizer (pré amostrar) o valor do tempo de serviço do próximo cliente e utilizar esse valor como *lookahead*. (Nicol, 1988; Wagner & Lazowska, 1989) definem também o cálculo do *lookahead* para outras disciplinas de fila.

Já o número de *rollbacks* pode ser calculado analisando o modelo a ser simulado, sua parametrização e a plataforma onde será executado. Existem várias características que podem influenciar a quantidade de *rollbacks*, tais como:

- Probabilidade de um cliente voltar ao sistema (realimentação);
- Plataforma onde a simulação será executada (se esta é homogênea ou não);

A probabilidade de um cliente voltar ao sistema influencia fortemente o número de *rollbacks* devido à própria definição de quando um *rollback* pode acontecer. Um *rollback* deve acontecer quando são detectados erros de causa e efeito. Esses erros ocorrem quando, por exemplo, um evento e_2 , que deveria ser tratado antes de e_3 , é tratado depois (seção 3.2). Em sistemas que possuem realimentação, quanto maior a probabilidade dos clientes voltarem ao sistema, maior será o número de *rollbacks*.

Outro fator que influencia o número de *rollbacks* é a plataforma onde a simulação será executada. As plataformas homogêneas³ não influenciam o número de *rollbacks*. Já nas plataformas heterogêneas o número de *rollbacks* pode ser alto ou não, dependendo de como

³ Entende-se por plataforma homogênea aquela em que as máquinas possuem características bem semelhantes

os processos lógicos foram mapeados. Caso haja a necessidade de mapear a realimentação em um processo lógico diferente, estando estes em máquinas distintas e que possuem poder computacional diferente, e o processo lógico que vai receber os clientes da realimentação estiver mapeado em uma máquina mais rápida, então a possibilidade de ocorrer *rollback* é maior.

A próxima seção ilustra como as conclusões descritas nesta seção e na anterior podem ser utilizadas para a implementação do módulo Avaliador, níveis 2 e 3 respectivamente.

4.4.4 Utilização de Inteligência Artificial no Módulo Avaliador

Uma maneira de implementar o módulo avaliador, permitindo que as informações utilizadas na decisão possam ser adequadas a cada ambiente em que o ASDA for utilizado, é através da utilização de conceitos da área de Inteligência Artificial ou Inteligência Computacional.

Um sistema de Inteligência Artificial (IA) é capaz de adquirir, representar e manipular conhecimento. A manipulação inclui a capacidade de deduzir ou inferir novos conhecimentos (novas relações sobre fatos e conceitos) a partir do conhecimento existente.

Todo programa de computador possui um determinado conhecimento sobre um problema a ser resolvido. O conhecimento está no algoritmo que o programa emprega e nas rotinas de tomada de decisão. No entanto, as informações estão representadas implicitamente e não são fáceis de serem atualizada ou manipuladas. Uma das principais características de IA é que o sistema é estruturado de modo a separar os algoritmos de decisão (conhecimento procedimental) dos fatos e regras (conhecimento declarativo).

Um sistema de aprendizagem é um programa que toma decisões baseado num conjunto de casos bem sucedidos. Dessa forma, o principal objetivo de um sistema de aprendizagem é classificar com sucesso novos casos. A figura 4.12 ilustra a estrutura básica de um sistema de classificação que aceita como entrada um padrão de dados e produz como saída uma decisão (Weiss & Kulikowski, 1991).

Cada caso bem sucedido que faz parte do padrão de dados é composto de observações (também referenciadas como características) e a classe correta a que o caso está associada.

Figura 4.12. Estrutura básica de um sistema classificador

Tendo o sistema de aprendizagem o objetivo de classificar com sucesso novos casos, deve haver um meio de se verificar se a classificação está tendo sucesso ou está falhando. A maneira mais utilizada é a taxa de erros, definida como:

$$\text{taxa de erro} = \frac{\text{número de erros}}{\text{número de casos}}$$

Existem vários métodos que podem ser utilizados no problema de classificação, os quais são independentes de qualquer aplicação específica. Dentre esses métodos, pode-se citar: reconhecimento de padrão estatístico (classificadores Bayesianos, discriminantes lineares, métodos do vizinho mais próximo), redes neurais e máquinas de aprendizagem (árvores de decisão e regras de indução).

Uma das alternativas para a utilização de sistemas de aprendizagem é o método do vizinho mais próximo. Este método consiste em verificar diretamente a tabela que contém os casos e tentar associar a um padrão na tabela de casos. Caso não ocorra a associação com um padrão específico, o classificador busca o padrão mais próximo e classifica o caso na classe desse padrão. Este método é simples e possui uma taxa de erro não muito alta (Weiss & Kulikowski, 1991). Uma tabela que contenha as características e a classe associada a cada conjunto de características é necessária neste método bem como um algoritmo que faça a busca nessa tabela.

Um exemplo de como esse método pode ser utilizado no nível 2 do módulo avaliador pode ser observado na tabela 4.3, onde uma parte das possibilidades são descritas e tem-se como características:

- Tamanho do modelo (TM): pequeno, médio, grande;
- Tamanho do warm-up (TWU): pequeno, médio, longo;
- Precisão (PR): pequena, média, alta;
- Paralelismo implícito no modelo (PI): pequeno, médio, alto;
- Plataforma onde a simulação será executada (PL): tipo 1, tipo 2, tipo 3;
- Número de processadores (NP): pequeno, médio, alto;

e como classes:

- SRIP
- MRIP

Quando uma característica tem uma influência maior sobre as outras, esta é predominante e as outras características não precisam ser analisadas (denotadas por X).

Tabela 4.3. Associação entre as características e a classe para o nível 2 do módulo Avaliador

TM	TWU	PR	PI	PL	NP	Classe
Pequeno	Grande	X	Alto	Tipo 2	X	SRIP
Médio	Grande	X	Alto	Tipo 2	X	SRIP
X	X	X	Baixo	Tipo 3	X	MRIP
X	X	X	Médio	Tipo 3	X	MRIP
Grande	X	X	X	Tipo 2	Baixo	SRIP
Grande	X	X	X	Tipo 2	Médio	SRIP
Médio	X	X	X	Tipo 3	Alto	MRIP
Médio	X	X	X	Tipo 3	Médio	MRIP
Médio	X	X	X	Tipo 3	Alto	MRIP
Médio	X	X	X	Tipo 3	Médio	MRIP
Pequeno	X	X	X	Tipo 3	Alto	MRIP
Pequeno	X	X	X	Tipo 3	Médio	MRIP

A tabela 4.4 ilustra uma parte das possibilidades para a utilização desta mesma técnica para o nível 3 do módulo Avaliador. As características definidas são:

- *Lookahead* (LA): pequeno, médio, alto;
- Realimentação entre processos lógicos (RE): pouca, média, alta;
- Plataforma (PF): homogênea, heterogênea;

e as classes:

- CMB
- TW (Time Warp)
- $CMB \Rightarrow TW$

Tabela 4.4. Associação entre as características e a classe para o nível 3 do módulo Avaliador

LA	RE	PF	Classe
Baixo	Alto	Heterogênea	$CMB \Rightarrow TW$
Alto	Alto	X	CMB
Médio	Baixo	Homogênea	TW

Como pode ser observado, definir as tabelas 4.3 e 4.4 não é uma tarefa simples, pois muitas são as possibilidades. Por exemplo, para a tabela do nível 2 são: 6 variáveis com 3 possibilidades para cada uma o que resulta em 3^6 combinações. Visando buscar uma forma mais viável de realizar a tarefa de tomar decisões, surgiu o conceito de lógica nebulosa, também conhecida como Lógica Fuzzy. A lógica fuzzy insere-se também na Inteligência Artificial e foi desenvolvida para prover uma maneira de se conseguir conclusões a partir de informações vagas, ambíguas ou imprecisas, aproximando o computador da maneira de pensar humana, a qual consegue achar soluções precisas a partir de dados aproximados.

Diferentemente da lógica clássica, na qual é possível representar somente os estados 0 e 1, ou verdadeiro e falso, ou sim e não, etc., a lógica fuzzy permite que sejam representados continuamente vários estados, tais como as várias tonalidades existentes entre as cores branca e preta.

A figura 4.13 exemplifica um gráfico gerado pelo *toolbox* de lógica fuzzy do MatLab para a distribuição proposta na figura 4.5 para a escolha entre a utilização das abordagens MRIP e SRIP. A parte do gráfico em preto representa a escolha da abordagem MRIP (plataformas com bom processamento e pouca comunicação). A parte em cinza escuro representa a escolha da abordagem SRIP (plataformas com processamento não muito elevado e boa comunicação) e a parte em cinza claro representa as plataformas que podem executar tanto a abordagem SRIP quanto a abordagem MRIP.

Figura 4.13. Gráfico gerado pelo *toolbox* de lógica fuzzy do software Matlab para a escolha entre as abordagens MRIP e SRIP

Para gerar o gráfico da figura 4.13 foi necessário definir as funções de pertinência para a variável de entrada comunicação (figura 4.14) e para a variável de entrada processamento (figura 4.15). Juntamente com as regras, as funções de pertinência determinam o comportamento do sistema. As regras definidas para esta situação foram:

Figura 4.14. Função de pertinência para a variável comunicação

Figura 4.15. Função de pertinência para a variável processamento

1. If [Processamento is Lento] and [Comunicação is CL] then [Abordagem is MRIP/SRIP]
2. If [Processamento is Lento] and [Comunicação is CM] then [Abordagem is SRIP]
3. If [Processamento is Lento] and [Comunicação is CA] then [Abordagem is SRIP]
4. If [Processamento is Médio] and [Comunicação is CL] then [Abordagem is MRIP]
5. If [Processamento is Médio] and [Comunicação is CM] then [Abordagem is MRIP/SRIP]

6. If [Processamento is Médio] and [Comunicação is CA] then [Abordagem is SRIP]
7. If [Processamento is Rápido] and [Comunicação is CL] then [Abordagem is MRIP]
8. If [Processamento is Rápido] and [Comunicação is CM] then [Abordagem is MRIP]
9. If [Processamento is Rápido] and [Comunicação is CA] then [Abordagem is MRIP/SRIP]

sendo que CL = comunicação lenta, CM = comunicação média e CA: comunicação alta.

Para a definição de todo o nível 3 do módulo Avaliador utilizando lógica fuzzy é necessária a definição das outras variáveis e de novas regras, as quais fornecerão o resultado final ao usuário, ou seja, a escolha da abordagem.

4.4.5 Especificação do Módulo Avaliador em UML

O caso de uso “Utilizar o auxílio do ambiente” definido no Diagrama de Casos de Uso do ASDA apresentado na figura 4.4 formaliza o comportamento deste módulo e é descrito a seguir.

Caso de Uso: Utilizar o auxílio do ambiente

Descrição: Permite ao usuário utilizar o conhecimento que o ambiente possui para definir se vai utilizar uma simulação SRIP ou MRIP. Permite também utilizar o conhecimento para decidir entre a utilização dos protocolos Time Warp ou CMB e em como particionar o modelo em processos lógicos.

Pré-condição: Ter permissão para ser UsuarioSimulacao e estar desenvolvendo uma nova simulação.

Fluxo Principal de Eventos:

Ação do Ator	Resposta do Sistema
1. O usuário decide obter auxilio para o desenvolvimento de uma nova simulação	
	2. Verifica se o usuário precisa de auxílio para: <ol style="list-style-type: none"> a. Escolher entre utilizar MRIP ou SRIP b. Escolher entre utilizar os protocolos CMB, Time Warp ou CMB/Time Warp

	<p>c. Escolher como particionar o modelo em processos lógicos seguindo o protocolo de sincronização escolhido</p> <p>3. Caso o auxílio seja para escolher entre utilizar MRIP ou SRIP, verifica as características do modelo, a plataforma onde a simulação será executada e os objetivos e decide entre SRIP ou MRIP</p> <p>4. Caso o auxílio seja para escolher entre utilizar os protocolos CMB, Time Warp, CMB/Time Warp, verifica o lookahead, as características do modelo e o provável número de rollbacks e decide entre CMB, Time Warp ou CMB/Time Warp</p> <p>5. Caso o auxílio seja para particionar o modelo em processos lógicos seguindo o protocolo de sincronização escolhido verifica o grafo do modelo e determina a partição que resultará no melhor otimismo para o protocolo CMB ou no menor número de rollbacks no protocolo Time Warp</p>
--	--

Pós-condição: Ter proporcionado o auxílio desejado pelo usuário

4.5 Módulo Gerador

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

Este módulo divide-se em dois sub-módulos:

- Geração de código de programas MRIP;
- Geração de código de programas SRIP;

No ASDA, a decisão de qual abordagem utilizar pode ficar a cargo do usuário ou ser uma tarefa realizada pelo módulo gerador com o auxílio do módulo Avaliador, nível 2. O funcionamento do módulo avaliador nível 2 é descrito na subsecção 4.4.2.

O ambiente deve permitir a geração de código em algumas linguagens/extensões funcionais pré-determinadas. Para cada linguagem/extensão deve haver dois arquivos (figura 4.16):

- 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.

Figura 4.16. Estrutura de arquivos no módulo Gerador

Um exemplo de um arquivo `gabarito.dat` pode ser observado na figura 4.17 (Spolon, 1994a). Este arquivo é um esqueleto de um programa de simulação sequencial que utiliza a extensão funcional SMPL (MacDougall, 1987).

```

1  /* ----- */
2  #include <dir.h>
3  #include <stdio.h>
4  #include <conio.h>
5  #include "smpl.h"
6  #include "rand.h"
7
8  main()
9  {
10     /* Fase de Definições */
11     %0define_tempo_num_max
12     int Event = 1, Customer = 1, Aleatorio;
13     %Agera_contadores_loop
14     %1define_variaveis
15
16     /* Fase de Preparação */
17     %2define_smpl
18
19     /* cria e da nome as facilidades */
20     %3define_server
21
22     /* escalona a chegada do primeiro cliente */
23     %4escalona_primeiro_evento
24
25     %Bgera_trace
26
27     /* Fase de Implementação */
  
```

```

27 %5gera_loop
28 {
29     %Gwarm_up
30     %6gera_cause
31     %7gera_switch
32     {
33         %8gera_eventos
34     }
35 }
36 /* gera o relatorio da simulacao */
37 %9gera_relatorio_padrao
38 %Cgera_relatorio_estatisticas_max_min
39 %Dgera_relatorio_estatisticas_fila_vazia
40 }
41 /* ----- */

```

Figura 4.17. Arquivo gabarito.dat para a extensão funcional SMPL

Como pode ser observado, este gabarito é dividido em três fases (Spolon, 1994a):

- **Fase de Definição:** geração da estrutura de dados para a execução do modelo;
- **Fase de Preparação:** geração de primitivas de inicialização do ambiente de simulação;
- **Fase de Implementação:** geração dos eventos que compõem o programa de simulação.

A geração do código para a abordagem MRIP nada mais é do que a geração de um programa de simulação sequencial com o acréscimo de uma biblioteca com funções de comunicação (para envio dos dados para o mestre) e uma biblioteca com funções para o tratamento local das observações coletadas antes de enviar para o mestre. Na fase de especificação gráfica já foram definidos qual(uais) centro(s) de serviço e qual(uais) o(s) parâmetro(s) devem ser analisado(s). O módulo Gerador obtém essas informações para que a função de coleta da observação do parâmetro desejado seja inserida no local correto.

Para a geração de um programa MRIP é necessária a inclusão dos seguintes trechos no gabarito (figura 4.18):

- Fase de Definição:
 - bibliotecas (linhas 6, 7, 8 e 9);
 - variáveis para comunicação (linhas 16 e 17);
- Fase de Preparação:
 - funções para comunicação (linha 22);
 - inicialização do método de análise dos resultados (linha 23);
 - inicialização do gerador de números aleatórios (linha 24)
- Fase de Implementação:
 - dentro da função que gera os eventos (linha 42), deve ser incluída a função

de observação do(s) parâmetro(s) escolhido(s). Essa função envia o valor observado para o mestre que se encarrega de verificar se a simulação já pode ser encerrada ou não.

```

1  /* ----- */
2  #include <dir.h>
3  #include <stdio.h>
4  #include <conio.h>
5  #include "smplx.h"
6  #include "communication.h"
7  #include "debug.h"
8  #include "distributions.h"
9  %0include_metodo_analise
10
11 main()
12 {
13     /* Fase de Definição */
14     %1define_tempo_num_max
15     int Event = 1, Customer = 1, Aleatorio;
16     int me, id;
17     char *host_name;
18     %Agera_contadores_loop
19     %2define_variaveis
20
21     /* Fase de Preparação */
22     %3inicializa_comunicação
23     %4inicializa_metodo_analise
24     %5inicializa_GNA
25     %6define_smpl
26
27     /* cria e da nome as facilidades */
28     %7define_server
29
30     /* escalona a chegada do primeiro cliente */
31     %8escalona_primeiro_evento
32
33     %Bgera_trace
34
35     /* Fase de Implementação */
36     %9gera_loop
37     {
38         %Gwarm_up
39         %10gera_cause
40         %11gera_switch
41         {
42             %12gera_eventos
43         }
44     }
45     /* gera o relatorio da simulacao */
46     %13gera_relatorio_padrao
47     %Cgera_relatorio_estatisticas_max_min
48     %Dgera_relatorio_estatisticas_fila_vazia
49 }
50 /* ----- */

```

Figura 4.18. Gabarito para geração de código MRIP

A abordagem SRIP pode utilizar os protocolos conservativo, otimista ou a troca de protocolos em tempo de execução. Desta forma, a geração de código para a abordagem SRIP pode seguir três caminhos:

- Geração de código para programas que utilizam protocolo conservativo CMB;

- Geração de código para programas que utilizam protocolo otimista Time Warp;
- Geração de código para programas que utilizam a troca de protocolos em tempo de execução.

Assim como a escolha entre os protocolos SRIP e MRIP pode ser realizada automaticamente pelo ASDA, a escolha entre a utilização entre os protocolos SRIP também pode contar com o auxílio do módulo Avaliador, nível 3 para a escolha entre um protocolo ou outro. Além disso, esse nível do módulo avaliador é também responsável pela divisão do modelo em processos lógicos, como detalhado na seção anterior.

Independente de qual protocolo é utilizado, a geração do código SRIP é feita da mesma maneira que no caso da abordagem MRIP. Para cada linguagem de simulação ou extensão funcional disponível no ASDA, deve haver um arquivo `gabarito.dat` e um programa que lê este gabarito e os dados do modelo e gera o programa de simulação. A diferença é que no caso de MRIP gera-se somente um programa enquanto que em SRIP são gerados tantos programas quanto o número de processos lógicos, ou seja, para cada processo lógico existe um programa diferente. Porém, o esqueleto de todos os programas é o mesmo.

Gerado o programa, o módulo Gerador atualiza o arquivo do usuário que contém as informações referentes aos modelos do usuário, definido na subseção 4.3.1.

Caso o usuário escolha realizar uma simulação sequencial ou SRIP do tipo horizonte infinito e a linguagem/extensão não possua nenhum método para decidir quando a simulação deve ser encerrada, o programa gerador insere a função de observação da abordagem MRIP e passa a tratar esse programa como sendo um programa MRIP ou (SRIP + MRIP). Neste caso, o número de replicações é igual a um (1) de modo a permitir utilizar os métodos de análise de saída implementados no módulo replicador. Esta opção não afeta o desempenho da simulação pois o programa mestre (para coleta dos resultados) e o programa de simulação podem ser escalonados para ser executados na mesma máquina, não sendo necessária a utilização da rede.

Para exemplificar a geração de uma simulação SRIP, a figura 4.19 apresenta um arquivo `gabarito.dat` para a extensão funcional ParSMPL (Ulson, 1999), que é uma extensão do SMPL e implementa o protocolo de sincronização CMB. As linhas em negrito e itálico representam trechos de código que foram acrescentados a partir do gabarito original para SMPL e que devem ser inseridos tanto no programa mestre como nos programas escravos. As linhas em itálico devem ser acrescentadas somente no programa mestre e as linhas em negrito somente nos programas escravo.

```
1 /* ----- */
2 #include <dir.h>
3 #include <stdio.h>
4 #include <conio.h>
5 #include <pvm3.h>
6 #include "declaracoes.h"
```

```

7  #include "parsmpl.h"
8
9  %0define_recurso
10 %Apai_define_SLAVENAME
11
12 void main(void)
13 {
14     /* Fase de Definição */
15     %1define_tempo_num_max
16     int Event = 1, Aleatorio;
17     int tid_processo_pai;
18
19     %Bgera_contadores_loop
20     %2define_variaveis
21
22     /* Fase de Preparação */
23
24     %Cpai_registra_pvm
25     %Dpai_prepara_tabelas
26     %Epai_envia_tabela
27     %Fescravo_inicia_tids
28     %Gescravo_recebe_tabela_tid
29
30     %3inicia_clock_cana
31     %4define_smp1
32
33     /* cria e da nome as facilidades */
34     %5define_server
35
36     /* escalona a chegada do primeiro cliente */
37     %Hpai_escalona_primeiro_evento
38
39     %Igera_trace
40
41     /* Fase de Implementação */
42     %6gera_loop
43     {
44         %Lwarm_up
45         %7gera_cause
46         %8gera_switch
47         {
48             %9gera_eventos
49         }
50     }
51     /* gera o relatorio da simulacao */
52     %10gera_relatorio_padrao
53     %Jgera_relatorio_estatisticas_max_min
54     %Kgera_relatorio_estatisticas_fila_vazia
55     parsmpl_pvm_exit ();
56 }
57 /* ----- */

```

Figura 4.19. Gabarito para geração de código SRIP, utilizando a extensão ParSMPL

Após os arquivos terem sido gerados, é necessário que sejam compilados e enviados para o módulo Execução, que será descrito na subseção 4.7.

4.5.1 Especificação do Módulo Gerador em UML

Os casos de uso “Definir uma simulação SRIP” e “Definir uma simulação MRIP” e suas extensões definidos no Diagrama de Casos de Uso do ADSA apresentado na figura 4.4 formalizam o comportamento descrito nesta seção e são descritos a seguir.

Caso de Uso: Definir uma simulação MRIP

Descrição: Permite ao usuário a possibilidade de desenvolver uma simulação seguindo a abordagem MRIP.

Pré-condição: Ter permissão para ser *UsuarioSimulacao*; possuir um sistema a ser simulado, e este estar modelado seguindo as abordagens *Redes de Filas* ou *Queueing Statecharts*.

Fluxo Principal de Eventos:

Ação do Ator	Resposta do Sistema
1. O usuário solicitou o auxílio do ambiente e este optou por gerar um programa de simulação MRIP ou o próprio usuário optou por gerar um programa de simulação MRIP	
	2. Gera o programa de simulação baseado na especificação gráfica, nos parâmetros fornecidos pelo usuário e nos parâmetros do módulo replicador
	3. Compila e gera os executáveis
	4. Inicia a execução
	5. Exibe os resultados

Fluxo Alternativo de Eventos:

Linha 2: O sistema não consegue gerar o programa de simulação

Linha 3: O novo arquivo gerado possui problemas de compilação ou linkedição

Pós-condição: Possuir os resultados fornecidos pela execução do programa de simulação utilizando a abordagem MRIP

Caso de Uso: Definir uma simulação SRIP

Descrição: Permite ao usuário a possibilidade de desenvolver um programa de simulação seguindo a abordagem SRIP.

Pré-condição: Ter permissão para ser UsuarioSimulacao e possuir um sistema a ser simulado já especificado.

Fluxo Principal de Eventos:

Ação do Ator	Resposta do Sistema
1. O usuário solicitou o auxílio do ambiente e este optou por gerar um programa de simulação SRIP ou o próprio usuário optou por gerar um programa de simulação SRIP	
	2. Fornece tela perguntando ao usuário se ele deseja escolher o protocolo de sincronização ou se ele deseja que o sistema escolha automaticamente
3. Escolhe entre os protocolos de sincronização ou deixar o sistema escolher	

Fluxo Alternativo de Eventos:

Linha 3: O usuário cancela a operação

Extend:

Linha 3: Utilizar o auxílio do ambiente

Pós-condição: Ter escolhido um protocolo de sincronização para geração do programa de simulação utilizando a abordagem SRIP.

Caso de Uso: Definir uma simulação SRIP/CMB

Descrição: Permite ao usuário a possibilidade de desenvolver uma simulação seguindo a abordagem SRIP utilizando o protocolo CMB.

Pré-condição: Ter permissão para ser UsuarioSimulacao e possuir uma linguagem/biblioteca para simulação utilizando o protocolo conservativo CMB.

Fluxo Principal de Eventos:

Ação do Ator	Resposta do Sistema
1. O usuário solicitou o auxílio do ambiente e este optou por utilizar o	

protocolo CMB ou o próprio usuário optou por utilizar esse protocolo	
	2. Fornece tela perguntando ao usuário se ele deseja escolher a partição do modelo em processos lógicos ou se ele deseja que o sistema faça automaticamente
3. Escolhe entre particionar o modelo ou deixar o sistema particionar	
4. Se o usuário optou por particionar o modelo, especifica a partição	
	5. Se o usuário optou pelo sistema particionar o modelo em processos lógicos, especifica a partição 6. Gera os programas de simulação 7. Compila e gera os executáveis 8. Inicia a execução 9. Exibe os resultados

Fluxo Alternativo de Eventos:

Linha 3: O usuário cancela a operação

Linha 4: O usuário cancela a operação

Linha 6: O sistema não consegue gerar o programa de simulação

Linha 7: O novo arquivo gerado possui problemas de compilação ou linkedição

Pós-condição: Possuir os resultados fornecidos pela execução do programa de simulação utilizando a abordagem SRIP e o protocolo de sincronização CMB

Caso de Uso: Definir uma simulação SRIP/Time Warp

Descrição: Permite ao usuário a possibilidade de desenvolver uma simulação seguindo a abordagem SRIP utilizando o protocolo otimista Time Warp.

Pré-condição: Ter permissão para ser UsuarioSimulacao e possuir uma linguagem/biblioteca para simulação utilizando o protocolo otimista Time Warp.

Fluxo Principal de Eventos:

Ação do Ator	Resposta do Sistema
1. O usuário solicitou o auxílio do ambiente e este optou por utilizar o	

protocolo Time Warp ou o próprio usuário optou por utilizar esse protocolo	
	2. Fornece tela perguntando ao usuário se ele deseja escolher a partição do modelo em processos lógicos ou se ele deseja que o sistema faça automaticamente
3. Escolhe entre particionar o modelo ou deixar o sistema particionar 4. Se o usuário optou por particionar o modelo, especifica a partição	
	5. Se o usuário optou pelo sistema particionar o modelo em processos lógicos, especifica a partição 6. Gera os programas de simulação 7. Compila e gera os executáveis 8. Inicia a execução 9. Exibe os resultados

Fluxo Alternativo de Eventos:

Linha 3: O usuário cancela a operação

Linha 4: O usuário cancela a operação

Linha 6: O sistema não consegue gerar o programa de simulação

Linha 7: O novo arquivo gerado possui problemas de compilação ou linkedição

Pós-condição: Possuir os resultados fornecidos pela execução do programa de simulação utilizando a abordagem SRIP e o protocolo de sincronização Time Warp.

Caso de Uso: Definir uma simulação SRIP/CMB->TimeWarp

Descrição: Permite ao usuário a possibilidade de desenvolver uma simulação seguindo a abordagem SRIP utilizando o protocolo conservativo CMB com troca para o protocolo otimista Time Warp em tempo de execução.

Pré-condição: Ter permissão para ser UsuarioSimulacao e possuir uma linguagem/biblioteca para simulação utilizando o protocolo conservativo CMB com troca para o protocolo otimista Time Warp em tempo de execução.

Fluxo Principal de Eventos:

Ação do Ator	Resposta do Sistema
--------------	---------------------

1. O usuário solicitou o auxílio do ambiente e este optou por utilizar o protocolo CMB com troca para o protocolo Time Warp em tempo de execução ou o próprio usuário optou por utilizar esse protocolo	
	2. Fornece tela perguntando ao usuário se ele deseja escolher a partição do modelo em processos lógicos ou se ele deseja que o sistema faça automaticamente
3. Escolhe entre particionar o modelo ou deixar o sistema particionar	
4. Se o usuário optou por particionar o modelo, especifica a partição	
	5. Se o usuário optou pelo sistema particionar o modelo em processos lógicos, especifica a partição 6. Gera os programas de simulação 7. Compila e gera os executáveis 8. Inicia a execução 9. Exibe os resultados

Fluxo Alternativo de Eventos:

Linha 3: O usuário cancela a operação

Linha 4: O usuário cancela a operação

Linha 6: O sistema não consegue gerar o programa de simulação

Linha 7: O novo arquivo gerado possui problemas de compilação ou linkedição

Pós-condição: Possuir os resultados fornecidos pela execução do programa de simulação utilizando a abordagem SRIP e o protocolo de sincronização CMB com troca para o protocolo Time Warp em tempo de execução.

4.6 Módulo de Interface de Software

O Módulo de Interface de Software é ativado quando o usuário seleciona a opção de executar um programa de simulação já elaborado, utilizando a abordagem *MRIP*. A inclusão desse módulo no ASDA foi motivada pelo grande número de programas de simulação sequenciais já desenvolvidos. Além da execução sequencial, outro fator que torna a obtenção dos resultados ainda mais demorada é o fato de serem necessárias várias execuções do mesmo

programa (cada uma com uma semente de números aleatórios) para que uma boa precisão seja atingida. Permitir que esses programas sejam executados utilizando a abordagem MRIP diminui o tempo da simulação pois, além de diminuir o tempo de execução para a obtenção dos resultados confiáveis, não é necessário o desenvolvimento de um novo programa.

Deste modo, a idéia do módulo de interface de *software* é fornecer uma interface para cada *software* de simulação, tais como as extensões funcionais SMPL ou ParSMPL; os ambientes Warped, entre outros. Cada interface deve inserir no programa de simulação alvo os seguintes trechos de código:

- Nomes das bibliotecas (de comunicação e análise dos resultados);
- Variáveis;
- Inicialização do método de análise de saída;
- Inicialização da comunicação com o mestre;
- Funções para depuração;
- Função de coleta dos parâmetros a serem analisados

Para inserir esses trechos de código nos programas de simulação é necessário que o código fonte esteja disponível para que este possa ser percorrido e cada item acima seja inserido corretamente.

Após o programa ser modificado, ele deve ser compilado e encaminhado para o módulo execução, o qual é descrito na próxima subseção.

Caso de Uso: Utilizar um programa de simulação já desenvolvido

Descrição: Permite ao usuário a utilização de um programa de simulação já desenvolvido em outra linguagem, replicando-o para obter os resultados confiáveis com uma velocidade maior.

Pré-condição: Ter permissão para ser UsuarioSimulação; possuir um arquivo com o código do programa de simulação; o sistema possuir interface com o software de simulação utilizado pelo programa.

Fluxo Principal de Eventos:

Ação do Ator	Resposta do Sistema
1. O usuário seleciona a opção de replicar um programa de simulação já desenvolvido	
	2. Fornece a tela para o usuário indicar o arquivo e a linguagem/biblioteca de simulação utilizada

3. Seleciona o arquivo com o programa de simulação e a linguagem/biblioteca	
	4. Gera um novo programa de simulação
	5. Compila e gera o executável
	6. Inicia a execução
	7. Exibe os resultados

Fluxo Alternativo de Eventos:

Linha 3: O usuário cancela a operação

Linha 4: O sistema não consegue gerar um novo programa de simulação

Linha 5: O novo arquivo gerado possui problemas de compilação ou linkedição

Pós-condição: Possuir os resultados fornecidos pela execução do programa de simulação

4.7 Módulo Executor

O módulo execução é o módulo encarregado de ler na base de dados referente aos usuários as informações referentes ao programa que o usuário está trabalhando e iniciar a execução do programa de simulação.

Caso seja uma simulação seqüencial, o módulo executor inicia a execução do programa já compilado e aguarda o final da simulação para obter os resultados. Normalmente os resultados são direcionados para um arquivo, o qual é exibido para o usuário.

Para os outros casos, antes de iniciar a execução do programa, é necessário verificar em quais máquinas o usuário deseja executar a simulação, comunicar o módulo escalonamento (a ser descrito na subseção 4.9) e aguardar o estabelecimento da máquina virtual.

Se for uma simulação SRIP, estabelecida a máquina virtual, o módulo executor verifica o nome do programa a ser executado e inicia sua execução. Do mesmo modo que uma simulação seqüencial, encerrada a simulação os resultados são apresentados ao usuário.

Já para simulações MRIP, o módulo executor passa a tarefa de executar a simulação para o módulo replicador, que gerenciará todas as replicações e passará o resultado obtido para o módulo executor exibir.

4.8 Módulo Replicador

O módulo Replicador é responsável por suportar a execução da simulação utilizando a

abordagem MRIP. Para que isso seja possível, deve haver dois componentes básicos: o **mestre MRIP** e o **escravo MRIP** (figura 4.20).

Figura 4.20. Relação entre os processos mestre e escravo na simulação MRIP

O escravo MRIP é o programa da simulação, que pode ter sido gerado pelo módulo Gerador ou pelo módulo Interface de Software. Como já descrito, esse programa contém as funções para comunicação com o mestre e também funções para tratamento local das observações coletadas.

O mestre MRIP é o que recebe as informações de todos os escravos e faz a análise estatística referente ao parâmetro observado. Se a variância das observações está dentro de um intervalo de confiança estabelecido por um nível de confiança, então a simulação é encerrada e os escravos são finalizados. Caso contrário, os escravos continuam sua execução até o ponto em que novas observações são encaminhadas ao mestre e o processo se repete.

Cada replicação é a execução de uma simulação sequencial. Deste modo, em cada replicação são executados dois estágios (figura 4.21): o estágio de detecção do período transiente e o estágio de estimação dos parâmetros (Pawlikowski, 1990).

Caso o usuário tenha optado por definir o tamanho do período transiente e/ou o tamanho do lote, o mestre MRIP ajusta as variáveis no escravo MRIP e inicia as replicações. Neste caso, corre-se o risco de o valor fornecido pelo usuário ser acima do valor real, desperdiçando tempo, ou ser abaixo do valor real, o que pode influenciar as observações do período de estimativa.

Caso contrário, o ambiente fica encarregado de determinar esses valores através de algumas funções. Para uma primeira aproximação do valor do tamanho do período transiente pode-se utilizar uma das dez heurísticas descritas em (Pawlikowski, 1990). Na fase de estimação dos parâmetros podem ser utilizados os métodos descritos em 2.4.6.

Figura 4.21. Dois estágios de cada replicação

4.9 Módulo Escalonamento

O principal objetivo do escalonamento é distribuir serviços (processos, tarefas, *threads*, etc.) aos vários recursos de processamento (processadores, memória, discos, etc.) disponíveis. Esta atividade pode ser realizada de diferentes maneiras, dependendo da política de escalonamento adotada, a qual depende dos objetivos definidos. O desempenho do escalonamento irá depender também dos seguintes fatores (Souza *et al.*, 1999):

- O objetivo da plataforma utilizada: o escalonamento adotado em uma arquitetura paralela, onde aplicações paralelas são executadas, difere do adotado em sistemas distribuídos que executam aplicações sequenciais;
- O *hardware* disponível: os recursos disponíveis para computação e comunicação;
- Ambientes multiusuário: vários usuários compartilhando os recursos computacionais podem provocar alterações no ambiente;
- Diferentes classes de aplicações: as aplicações podem ser classificadas sob diferentes pontos de vista: aplicações que requerem mais processamento (CPU-bound) ou mais comunicação (I/O-bound); quanto ao grau de interação com o usuário (interativas, *batch* ou tempo real); entre outras;
- Diferentes cargas de trabalho: ambientes com baixo ou alto índices de carga se comportam de maneira diferente utilizando a mesma política.

Santos em (Santos, 2001) fornece uma lista de alguns softwares de escalonamento atualmente disponíveis, tanto comerciais como acadêmicos:

- *LoadLeveler*: software comercial mantido pela *IBM*; voltado principalmente para máquinas *SP2*;

- *Load Sharing Facility*: software comercial mantido pela empresa *Platform*, pode ser utilizado por diferentes sistemas operacionais (SunOS, SGI IRIX, ULTRIX, ConvexOS) e diferentes ferramentas para desenvolvimento de aplicações paralelas (PVM, P4, Linda, HPF);
- *Condor*: software de domínio público desenvolvido por pesquisadores da Universidade de Wisconsin e do Laboratório Brookhaven. Seu desenvolvimento foi motivado pela possibilidade de utilizar a potência computacional de máquinas ociosas para a execução de aplicações dos usuários;
- *Codine (COMputing in Distributed Networked Environments)*: software comercial mantido pela empresa Genias Software e que pode ser utilizado em diferentes plataformas e sistemas operacionais tais como: ConvexOS, Cray, UNICOS, HP-UX, DEC OSF/1 e IBM AIX;
- *Load Balancer*: ferramenta para escalonamento de aplicações em uma rede de computadores que executam o sistema operacional UNIX;
- *AMIGO (DynAMical FlexIble SchedulinG EnvirOnment)*: software desenvolvido no Grupo de Sistemas Distribuídos e Programação Concorrente do ICMC-USP que tem por finalidade gerenciar o escalonamento em plataformas distribuídas de maneira flexível e dinâmica. O ambiente é flexível devido à capacidade de alterar as configurações dos fatores que influenciam o processo de escalonamento de acordo com as necessidades do usuário e é dinâmico devido à capacidade de adequar o ambiente com relação às novas configurações em tempo de execução.

Devido à importância do escalonamento em uma aplicação distribuída e sendo o ASDA um ambiente para execução de simulação distribuída, verificou-se a necessidade de incluir um módulo responsável pelo escalonamento dos programas de simulação.

O AMIGO foi o software escolhido para essa tarefa devido ao fato dele ter sido desenvolvido no mesmo grupo deste trabalho, o que facilita o acesso ao código e a solução das dúvidas.

Atualmente o AMIGO possui uma interface de comunicação com os ambientes de passagem de mensagem PVM (Araújo, 1999), MPI (Figueiredo, 2000) e Corba (Santos, 2001). Possui ainda uma política de escalonamento cpu-bound denominada DPWP (*Dynamic Policy Without Preemption*) (Araújo, 1999). A DPWP utiliza dados da potência computacional e também o número de processos na fila da CPU de cada máquina para fazer o escalonamento.

Para obter os dados da potência computacional é necessário, antes de iniciar a máquina virtual, executar 100 vezes uma aplicação e armazenar a média dos tempos em um arquivo na

própria máquina. Com base em todos os tempos médios da execução da aplicação deve-se verificar qual o menor tempo e fazer uma normalização para valores entre 0 e 1, fornecendo à máquina mais lenta o valor 1. Esse valor passa a ser a potência computacional da máquina e é armazenado em um arquivo na própria máquina.

Na estrutura do AMIGO, o primeiro *amigod* (AMIGO *daemon*) a ser executado estabelece um socket UDP para comunicação com outros *amigod*. O mesmo acontece com as políticas, como pode ser observado na figura 4.22.

Figura 4.22. Interação PVM e AMIGO

Ao adicionar uma máquina na máquina virtual, a DPWP que está sendo executada na máquina adicionada envia a potência computacional para a DPWP que ela tem comunicação. Com base nesses valores, a DPWP classifica as máquinas como: ociosa, moderada e carregada.

O PVM utiliza um vetor que possui os identificadores das máquinas para as quais os processos podem ser escalonados, sendo o escalonamento *round-robin*. Sem o AMIGO a ordem desse vetor é a ordem com que as máquinas são inseridas. Já com o AMIGO, a DPWP verifica os valores da potência de cada máquina e coloca na primeira posição a máquina com menor índice, ou seja, a que obteve um tempo menor na execução da aplicação. A DPWP verifica também as máquinas que estão classificadas como carregadas e evita enviar processos para essas máquinas.

Sendo assim, com a utilização do AMIGO e da DPWP, a função do módulo Escalonamento é executar automaticamente a aplicação nas máquinas que o usuário forneceu; coletar os valores médios, normalizando-os; e adicionar as máquinas à máquina virtual. Assim que outras políticas de escalonamento forem sendo inseridas no AMIGO, o módulo de escalonamento deve ir se adaptando de modo a suportar as necessidades para a utilização dessas novas políticas.

Esse processo permite que o usuário não precise se preocupar com o estabelecimento da máquina virtual e, mais importante, permite que o ASDA utilize todas as vantagens fornecidas pelo ambiente de escalonamento AMIGO.

Definidos os módulos do ASDA, a próxima seção utiliza mais dois diagramas da linguagem UML (o diagrama de Casos de Uso foi desenvolvido nas seções anteriores) com o objetivo de avançar um pouco mais na descrição formal do ambiente.

4.10 Diagramas UML

Com o intuito de formalizar melhor o comportamento do ASDA, esta seção apresenta os diagramas de classe e de atividades definidos para o ambiente.

4.10.1 Diagrama de Classes do ASDA

O diagrama de Classes é um diagrama estrutural que serve para visualizar, especificar, construir e documentar os aspectos estáticos do ambiente.

Uma visão dos módulos do diagrama de Classes do ASDA pode ser observado na figura 4.23. Nas figuras 4.24 e 4.25 podem ser observadas as expansões das classes de cada módulo.

Figura 4.23. Diagrama de classes do ASDA, visão dos módulos

Figura 4.24. Diagrama de classes do ASDA, módulo Interface e Especificação Gráfica

Figura 4.25. Diagrama de Classes do ASDA, módulo Geração dos Programas de Simulação

4.10.2 Diagrama de Atividades do ASDA

O diagrama de Atividades do ASDA foi desenvolvido baseado no diagrama de Casos de Uso e em sua especificação. A figura 4.26 ilustra este diagrama.

Figura 4.26. Diagrama de Atividades do ASDA

4.11 Comparação entre os ambientes descritos

Dentre os ambientes descritos no capítulo anterior observa-se que nenhum apresenta a possibilidade de desenvolvimento de uma simulação seguindo as abordagens SRIP ou MRIP dentro de um mesmo ambiente. Mais do que isso, nenhum ambiente descrito oferece ao usuário uma particularidade do ASDA: o auxílio na escolha da abordagem e protocolos de sincronização, escolha esta baseada nas características do modelo, na plataforma onde a simulação será executada e nos objetivos a serem alcançados.

Além disso, a maioria dos ambientes descritos não possui a característica de geração automática do código do programa de simulação onde o ambiente pode gerar um programa de simulação a partir de um modelo mais simples e o usuário pode adicionar funções mais elaboradas no próprio código do programa. Pode ainda simular diversas situações a partir de um único modelo, alterando somente a parametrização.

Desta forma, o ASDA procura fornecer ao usuário um ambiente que possua as principais características dos ambientes automáticos de simulação e também algumas características próprias e que tornam o ASDA um ambiente único.

4.12 Considerações Finais

Neste capítulo foi especificado o ASDA – Ambiente de Simulação Distribuída Automático. Com o objetivo de realizar esta especificação, foram utilizadas duas abordagens: uma informal e outra formal.

A especificação informal do ASDA foi realizada através de um diagrama modular e do detalhamento de cada um dos módulos:

- Módulo de Interface com o Usuário: responsável por toda a interação do usuário com o ambiente e responsável também por fornecer as ferramentas para o usuário especificar o modelo a ser simulado;
- Módulo Avaliador: responsável por auxiliar o usuário na especificação do modelo e na elaboração do programa de simulação, seguindo as abordagens sequencial, MRIP, SRIP (CMB, Time Warp ou CMB/Time Warp) e SRIP + MRIP;
- Módulo Gerador: responsável por gerar o programa de simulação com base no modelo especificado pelo usuário e seguindo a abordagem também especificada pelo usuário ou escolhida pelo próprio ambiente;
- Módulo de Interface de Software: utilizado quando o usuário deseja utilizar um programa de simulação já elaborado e deseja replicá-lo. Este módulo é responsável pela inserção de algumas rotinas com a finalidade de viabilizar a replicação;
- Módulo Executor: responsável por iniciar as execuções de todas as simulações,

exceto as replicações, as quais serão passadas para o módulo Replicador;

- Módulo Replicador: este módulo é responsável pela coordenação das replicações geradas quando a abordagem MRIP foi selecionada. É possível utilizar 3 métodos de análise de saída para a avaliação de quando a simulação deve ser encerrada.
- Módulo Escalonamento: responsável pelo escalonamento dos processos (programas de simulação).

A especificação formal do ASDA foi iniciada com a utilização de três diagramas propostos pela linguagem de modelagem UML:

- Diagrama de Casos de Uso: diagrama dinâmico do sistema, utilizado para identificar o comportamento do sistema em várias situações;
- Diagrama de Classes: diagrama estático do sistema, utilizado para representar as classes existentes no sistema;
- Diagrama de Atividades: diagrama dinâmico, auxilia o diagrama de casos de uso na identificação do fluxo do processamento interno.

Com base nessa especificação, foi desenvolvido um protótipo do ASDA, o qual será descrito no próximo capítulo.

Implementação do Protótipo do ASDA

5.1 Considerações Iniciais

No capítulo 4 foram discutidos e definidos os detalhes necessários para a especificação do Ambiente de Simulação Distribuída Automático (ASDA). Neste capítulo é descrito o desenvolvimento em termos de implementação de alguns módulos do ASDA, sendo então definido um protótipo do ambiente proposto.

Dos sete módulos que fazem parte do ASDA, três já estão praticamente implementados: Módulo de Interface com o Usuário, Módulo Replicador e Módulo de Interface de Software. Os outros 4 módulos ainda não estão implementados mas existem outros trabalhos de mestrado/doutorado que estão sendo desenvolvidos dentro do Grupo de Sistemas Distribuídos e Programação Concorrente e que têm como objetivo a integração com o ASDA. A seguir é apresentada uma relação de cada módulo com os trabalhos que estão sendo desenvolvidos:

- **Módulo de Interface com o Usuário:** este módulo está sendo implementado em um programa de mestrado, após ter tido suas principais características definidas neste trabalho. As principais decisões do projeto de implementação deste módulo, tais como a escolha da ferramenta, o projeto da interface gráfica e o projeto da base de dados a ser utilizada foram definidas nesta tese e estão descritas na próxima seção. Baseando-se nestas decisões, a aluna de mestrado Vera Lúcia Silva Camargo está terminando seu mestrado, o qual tem como objetivo a implementação do Módulo de Interface com o Usuário;
- **Módulo de Especificação Gráfica:** conforme descrito no capítulo 4, a especificação gráfica dos modelos a serem simulados deverá utilizar as abordagens *Queueing Statecharts* ou Redes de Filas Estendidas. A especificação *Queueing Statecharts* utilizada no ASDA como uma opção para o usuário definir o modelo é o resultado do trabalho de doutorado do aluno Carlos Renato Lisboa Francês (Francês, 2001). A escolha destas técnicas foi feita no decorrer do desenvolvimento deste trabalho e as justificativas no capítulo 4. A aluna de mestrado Thais Souza Aiza vem trabalhando no detalhamento e implementação deste módulo;
- **Módulo Gerador:** a aluna de mestrado Eliane Sayuri Tatsumi tem como projeto de mestrado a inserção da extensão funcional ParSMPL (que implementa o protocolo CMB) no ASDA. Este trabalho tem como objetivo implementar parte do Módulo Gerador e parte do Módulo Avaliador (Tatsumi, 2002). A extensão funcional ParSMPL foi um dos resultados da tese de doutorado de Roberta Spolon Ulson (Ulson, 1999);

- **Módulo Gerador:** o trabalho de doutorado de João Carlos Moraes Morselli Junior (Morselli, 2000) propõe um mecanismo para troca de protocolo de sincronização em tempo de execução. O trabalho da aluna de doutorado Célia Leiko Ogawa Kawabata (Kawabata, 2002) propõe uma maneira de implementação desse mecanismo utilizando agentes móveis e lógica *fuzzy* e será utilizado no ASDA quando o Módulo Avaliador ou o próprio usuário optarem pela utilização do protocolo CMB/Time Warp;
- **Módulo Avaliador:** as conclusões obtidas nas teses de doutorado de Roberta Spolon Ulson (Ulson, 1999) e Renata Spolon (Spolon, 2001) e na dissertação de mestrado de Mara Andréa Dota (Dota, 2001) forneceram a base para as decisões que devem ser tomadas no módulo Avaliador. O trabalho de (Dota, 2001) realizou ainda a instalação e configuração da linguagem Warped (que implementa o protocolo Time Warp) para o desenvolvimento de simulações utilizando redes de filas e que também será inserido no ASDA como uma opção para desenvolvimento de programas de simulação utilizando o protocolo Time Warp;
- **Módulo Avaliador:** o trabalho a ser desenvolvido pelo aluno de doutorado Edmilson Marmo Moreira (Moreira, 2002) enfoca o problema do tempo nos métodos de sincronização em simulação distribuída e fornecerá novas conclusões para o cálculo de *lookahead*;
- **Módulo Escalonamento:** o ambiente AMIGO, desenvolvido no trabalho de doutorado de Paulo Sérgio Lopes de Souza (Souza, 2000) e pelos alunos de mestrado Aletéia Araujo (Araujo, 1999), Tatiana Figueiredo (Figueiredo, 2000) e Ricardo Ribeiro dos Santos (Santos, 2001) oferece ao usuário a possibilidade de realizar escalonamento de carga utilizando os ambientes de passagem de mensagem PVM e MPI e a especificação CORBA e é a ferramenta a ser utilizada pelo ASDA em seu Módulo Escalonamento.

Os objetivos principais de desenvolvimento de um protótipo para o ASDA são:

1. Mostrar a viabilidade de se implementar um ambiente deste tipo;
2. Disponibilizar um núcleo do ambiente proposto com o intuito de definir as características principais do ASDA para que os demais trabalhos a serem desenvolvidos sigam as diretrizes definidas;
3. Oferecer uma especificação detalhada de como deve ser a interação entre o usuário e o ambiente e quais as funcionalidades devem estar disponíveis ao usuário;
4. Desenvolver os tópicos que ainda não foram trabalhados no Grupo de Sistemas Distribuídos e Programação Concorrente do ICMC – USP.

A escolha do ambiente computacional onde o protótipo será desenvolvido é o primeiro passo para essa implementação. Devido ao fato do ASDA utilizar alguns trabalhos já

desenvolvidos no sistema operacional Linux e também devido ao fato desse sistema operacional ter sua utilização aumentada a cada ano, este ambiente foi escolhido para ser a base do desenvolvimento. O Linux é executado sobre uma rede de computadores pessoais (PCs) disponível no laboratório do Grupo de Sistemas Distribuídos e Programação Concorrente – ICMC - USP. Dentre os trabalhos já desenvolvidos que utilizam o Linux pode-se citar: a extensão funcional ParSMPL (desenvolvida em Unix e facilmente utilizada no Linux), a biblioteca Warped e o ambiente para escalonamento AMIGO.

Considerando os objetivos citados, decidiu-se por implementar os módulos Interface com o Usuário, Replicador e Interface de Software. Esta decisão deve-se primeiramente pelo fato do Módulo Interface com o Usuário ser o responsável por toda a interação do usuário com o ambiente e segundo pelo fato de que o Módulo Replicador é o módulo menos trabalhado até o momento pelo alunos do Grupo de Sistemas Distribuídos e Programação Concorrente (LaSDPC) do ICMC – USP. Sendo o Módulo de Interface de Software diretamente ligado ao Módulo Replicador, este também foi desenvolvido neste protótipo.

A seguir, as implementações dos Módulos Interface com o Usuário, Replicador e Interface de Software são descritas.

5.2 Módulo de Interface com o Usuário

Conforme descrito no capítulo 4, o Módulo de Interface com o Usuário deve oferecer uma interface ao mesmo tempo amigável, permitindo fácil acesso a todas as ferramentas que um usuário menos experiente tenha necessidade de usar, e flexibilidade, permitindo que usuários mais experientes possam explorar todas as opções disponíveis.

Esta seção tem como objetivo descrever como o Módulo de Interface com o Usuário está sendo desenvolvido, enfatizando a escolha da ferramenta, as características da ferramenta escolhida, as telas e a base de dados já desenvolvidos.

5.2.1 A escolha da ferramenta

O primeiro passo para a implementação do protótipo deste Módulo é a definição de qual ferramenta é a mais adequada. É importante que a ferramenta escolhida atenda a dois requisitos básicos:

- Ofereça mecanismos para o desenvolvimento de uma interface gráfica amigável;
- Possibilite a manipulação de um banco de dados onde estarão armazenadas as informações dos usuários e do sistema

Dentre as possíveis ferramentas disponíveis que atendem os requisitos básicos estipulados acima, cabe ressaltar:

- Kylix, da Borland
- Java, da Sun

Enquanto as duas ferramentas citadas acima possuem os requisitos básicos para o desenvolvimento do ASDA, a diferença básica entre elas é que Java é uma linguagem enquanto Kylix é um ambiente RAD (*Rapid Application Development*). Em termos de desenvolvimento de softwares que possuem a característica do ASDA, Kylix possui uma vantagem sobre Java: a facilidade de desenvolver a interface com o usuário, justamente devido ao fato de ser uma ferramenta RAD. As ferramentas RAD possibilitam aos desenvolvedores despendem menos tempo com a produção de interfaces de suas aplicações, uma vez que não precisam escrever muitas linhas de código, interagindo diretamente com as API's do Windows ou Linux, para terem acesso às funcionalidades gráficas e visuais destes sistemas operacionais. Com isso os desenvolvedores têm um ganho de produtividade, dedicando-se mais aos códigos específicos da aplicação.

5.2.2 A ferramenta Kylix

O Kylix foi desenvolvido para o sistema operacional Linux e a empresa desenvolvedora é a Borland, a mesma do conhecido e bastante difundido ambiente Delphi (ambiente integrado de desenvolvimento do Pascal Objects, para sistemas operacionais Windows, em uso desde 1.995).

Avaliação de desempenho de equipes de desenvolvedores utilizando Kylix, comparada com o desempenho de equipes que usam somente o compilador gcc do Linux, tem demonstrado uma relação de custo até 50% menor para as equipes Kylix e uma relação de esforço de programação também 50% menor (Roetzheim, 2001).

O IDE (*Integrated Development Environment*) do Kylix permite que uma aplicação possa ser executada, sem modificações nos programas fonte, nos ambientes Windows e nos ambientes Linux. Os programas são facilmente recompilados em cada ambiente, facilitando o porte e a migração de *softwares* entre os mesmos (Delphi™ 6 no Windows e Kylix no Linux). O programa em código de máquina nativo é otimizado para as arquiteturas Intel, nos dois sistemas operacionais, tornando a execução do aplicativo bem mais rápida do que seria em um ambiente com interpretador de *bytecodes*, como é o caso de Java (Borland, 2001a).

O Kylix utiliza o conceito de reuso de componentes através do fornecimento de uma biblioteca de classes com componentes prontos, a CLX - *Component Library Cross Platform*, que por sua vez é uma ampliação da biblioteca Q-Troll, utilizada nos principais *desktops*⁴ Linux, Gnome e KDE. A CLX incorporou também componentes da biblioteca VCL (*Visual Components Library*) do antigo Delphi. A CLX tem hoje mais de 190 componentes

⁴ *desktops* - ambientes gráficos visuais para tornar o sistema operacional mais amigável ao usuário, com uso de ícones, janelas, etc.

visuais e não visuais prontos. A figura 5.1 ilustra os módulos do Kylix e Delphi.

Figura 5.1. Módulos do Kylix e Delphi

O ambiente de desenvolvimento do Kylix fornece todas as ferramentas para desenvolver, testar e implementar uma aplicação, com: reuso de objetos da CLX, ajuda on-line de programação, modelos de aplicações e uma coleção de ferramentas de projeto, simplificando a prototipagem e diminuindo a necessidade de codificar tudo, sob a forma de texto, dos programas-fonte, com isso diminuindo o tempo de desenvolvimento.

O IDE Kylix mantém as características principais da linguagem Pascal Objects, ou seja: herança simples entre as classes, herança múltipla entre interfaces, não permite estouro de valores inteiros, verificação de valores de vetores, tratamento de exceções, geração dos executáveis em Assembler nativo da arquitetura de hardware, programação modular, ligações dinâmicas ou estáticas de rotinas (*units*), dados fortemente tipados, suporte a *threads* e compilador nativo otimizado (Borland, 2001b).

Algumas características do Kylix que podem tornar-se interessantes para o ASDA como um todo são:

- Aceita Assembler no meio do código Pascal;
- Permite criação de biblioteca de objetos compartilhados (shared objects) reutilizáveis do Linux padrão, “.so’s”;
- Tem escalabilidade de banco de dados e são suportados diversos gerenciadores, tais como: (MySQL, Interbase, Oracle e DB2);
- Possui diretivas de compilação;
- Programação two-way: pelo IDE (utilizando a programação visual) ou escrevendo diretamente o texto do código fonte;
- Componentes para aplicações WEB e Apache;
- Versões: Open Edition (gratuita), Desktop e Server;

- Distribuição dos aplicativos desenvolvidos pelos utilizadores de Kylix poderá ser livre (licença GNU), atendendo uma forte tendência atual.

5.2.3 A Interface Gráfica do ASDA

A interface gráfica do ASDA tem duas funções: interagir com o superusuário e permitir a utilização do ambiente pelos usuários de simulação (especificando um novo modelo de simulação ou utilizando um programa de simulação já pronto). A partir do desenho do modelo definido pelo usuário (especificação gráfica) são gerados os programas de simulação.

Cada usuário tem acesso somente aos seus modelos e programas através de uso de senhas, sendo disponíveis visões do banco de dados por usuário.

A interface desenvolvida no Kylix permite ao superusuário:

- cadastro de novos usuários;
- exclusão ou alteração de dados dos usuários;
- visualização dos dados de todos os usuários;
- gerenciamento de senhas e login's do ASDA;
- cadastro de informações sobre o ambiente (configurações de hardware, tipos de técnicas de modelagem, abordagens de simulação distribuída, linguagens/extensões funcionais disponíveis no ASDA, tamanhos padrões de warm-up e tamanhos dos lotes) e também informações sobre os módulos avaliador, replicador, executor e escalonamento;
- gerenciamento de um repositório de programas de simulação;
- criação de um banco de dados de modelos de especificação gráfica.

A interface desenvolvida no Kylix permite ao usuário de simulação:

- especificar o modelo de simulação (especificação gráfica + parametrização);
- selecionar a abordagem do programa de simulação;
- solicitar auxílio do ambiente para decisões;
- gerar o programa de simulação;
- executar o programa;
- visualizar os resultados obtidos com a simulação

O desenvolvimento da interface gráfica do ASDA está sendo realizado através de várias telas (que no Kylix são *forms*), compostas por caixas de textos, campos de edição, menus e barras de ferramentas.

A tela apresentada na figura 5.2 é a primeira a ser apresentada ao usuário quando o protótipo do ASDA é iniciado. Nesta tela inicial o usuário deve fornecer sua identificação e senha para que o sistema possa carregar os arquivos de configuração do usuário que deseja utilizar o sistema. Após fornecer a identificação, é fornecida uma segunda tela de informação sobre as possíveis operações que o usuário pode fazer. Caso seja um usuário do tipo *super-usuário* a tela da figura 5.3.a é apresentada. Caso seja um usuário do tipo *usuário de simulação* é apresentada a tela da figura 5.3.b.

Figura 5.2. Tela de identificação do usuário

(a)

(b)

Figura 5.3. Tela inicial de entrada do ASDA para (a) super-usuário (b) usuário de simulação

Após a tela da figura 5.3.a, é apresentado ao super-usuário a tela da figura 5.4.a, onde estão disponíveis as opções para alterar as variáveis do ambiente e também manipular o arquivo com os cadastros dos usuários. Após a tela da figura 5.3.b, é apresentado ao usuário de simulação a tela da figura 5.4.b, onde estão disponíveis as opções de desenvolver um novo modelo e executar um programa de simulação já desenvolvido.

Figura 5.4. Tela de início para (a) super-usuário (b) usuário de simulação

Caso o usuário de simulação escolha a primeira opção (Especificar Modelo de Simulação), é apresentada a tela para definição do modelo, a qual pode ser observada na figura 5.5. Ela é composta pela janela de edição, onde os modelos são especificados, e pelas barras de ferramentas onde estão os ícones necessários para a construção dos modelos. Até o momento é possível especificar o modelo utilizando somente a abordagem Redes de Filas.

5.2.4 Base de Dados do ASDA

As versões anteriores ao Delphi 6 para Windows, diferentemente do Kylix, possuíam um componente para acesso a banco de dados bastante poderoso, mas muito lento: a BDE (*Borland Data Engine*). No Kylix isto foi suprimido, e criou-se uma nova interface de acesso a bancos de dados mais rápida e bem menor, a dbExpress, que pode acessar os dados criados e mantidos pelo Interbase nas aplicações *desktop*, do lado cliente, ou os dados de um servidor mantidos por grandes sistemas gerenciadores de banco de dados como Oracle e DB2.

O Interbase é um banco de dados Cliente/Servidor completo, com todos os recursos que caracterizam um gerenciador: rotinas de armazenamento, *triggers*, *views*, replicação e proteção de dados.

Inicialmente o Interbase era um gerenciador de banco de dados comercializado pela Borland. Desde agosto de 2000, o mesmo é disponibilizado pela empresa gratuitamente, com os códigos-fonte abertos, mesmo que o usuário não possua a licença dos produtos Delphi ou Kylix.

Figura 5.5. Tela de Especificação Gráfica do ASDA

A única desvantagem de se utilizar o Interbase, no caso do Kylix, é a não existência de uma interface amigável para o desenvolvedor criar o banco e os esquemas das tabelas, sendo que tudo deve ser feito no modo texto utilizando a linguagem SQL (*Structured Query Language*) e a interface ISQL (*Interactive SQL*) (Borland, 2001b).

Com base no diagrama de classes da especificação do ASDA, representado nas figuras 4.24 e 4.25, foram criados os esquemas das tabelas utilizadas no modelo relacional, através da interface ISQL.

No processo de transformação das classes em tabelas do modelo relacional de bancos de dados, novas entidades podem ser criadas principalmente para permitir a representação dos relacionamentos entre as classes.

Para a criação da estrutura do banco de dados do ASDA é necessário executar os seguintes comando no *prompt* do ISQL:

```
CREATE DATABASE '/ASDA-IG/ASDA-BD/asda.gdb';  
  
INPUT texto.esquema.txt;  
  
SHOW TABLE;
```

onde `asda.gdb` é o nome do arquivo a ser criado, `texto.esquema.txt` é o nome do arquivo onde estão os comandos de criação das tabelas e `show table` mostra a tabela já criada.

Um exemplo de uma classe definida na especificação e mapeada no banco de dados relacional pode ser observada na figura 5.6. Nesta tabela foram agrupadas as classes `SuperUsuario` e `UsuarioSimulacao`. O que distingue os dois tipos de usuário é o campo `TipoUsuario`.

Tabela de Usuários

```
create table Usuarios
(
  IdUsuario varchar(10) not null,
  NomeUsuario varchar (30),
  Senha varchar (7) not null,
  TipoUsuario char (1) not null,
  TecnicaModelagem varchar (15) not null,
  Linguagem varchar (10) not null,
  Abordagem varchar (10) not null,
  Tamanhowarm-up byte not null,
  TamanhoLote byte not null,
  primary key (IdUsuario)
)
```

Figura 5.6. Tabela Usuários definida para o ASDA

A implementação do módulo de Interface com o Usuário é o foco do trabalho de mestrado da aluna Vera Lúcia Silva Camargo, matriculada no programa de Pós-Graduação do Instituto de Ciências Matemáticas e de Computação – USP – São Carlos.

5.3 Módulo Replicador

Como descrito no capítulo 4, o ASDA foi especificado para ser um ambiente capaz de oferecer várias alternativas para o usuário desenvolver uma simulação distribuída, utilizando para isso as abordagens SRIP (*Single Replication in Parallel*) e MRIP (*Multiple Replication in Parallel*). Um ambiente denominado Akaroa (Ewing *et al.*, 1998), desenvolvido na Universidade de Canterbury, Nova Zelândia, pelos pesquisadores K. Pawlikowski, D. McNickle e G. Ewing, utiliza a abordagem MRIP e, pelas características apresentadas, foi escolhido para ser o ambiente capaz de fornecer ao usuário do ASDA a possibilidade de desenvolver simulações MRIP. As características do ambiente Akaroa bem como seu funcionamento são descritos na subseção 3.6.2.

No entanto, algumas características do Akaroa tornaram este ambiente incompatível com o ASDA. Desta forma, houve a necessidade de desenvolver uma versão mais simplificada do Akaroa. O Akaroa utiliza *sockets* para a comunicação entre o mestre e os escravos. Na especificação do ASDA definiu-se pela utilização do ambiente AMIGO no módulo de Escalonamento (seção 4.9). Como já descrito, o AMIGO possui interface com os ambiente de passagem de mensagem PVM e MPI, além da arquitetura CORBA. O fato do Akaroa utilizar *sockets* implicaria na alteração do mecanismo de comunicação deste, de modo a possibilitar a integração com o AMIGO. Além desse fato, houveram dificuldades referentes à compilação, pois o ambiente foi inicialmente desenvolvido para o UNIX e estava sendo compilado utilizando o Linux. Posteriormente foi disponibilizada uma versão para Linux, porém esta também apresentou problemas de compilação devido ao fato de que eram utilizadas muitas diretivas de compilação e as distribuições do Linux utilizadas eram diferentes. A falta de documentação referente à compilação em ambientes Linux foi também um dos fatores que nos levaram a desenvolver um novo software.

Este novo software para simulação baseado na abordagem MRIP e utilizando como base o Akaroa, é denominado ASDA-MRIP e foi desenvolvido já no ambiente Linux, utilizando o PVM para comunicação entre o mestre e os escravo e manteve algumas características do Akaroa, tais como os métodos pelo qual o período transiente é determinado, os métodos para a estimação dos parâmetros e também o gerador de números aleatórios. Devido à simplificação, o ASDA-MRIP foi implementado permitindo que somente uma variável possa ser analisada cada vez que a simulação é executada, ou seja, não se pode analisar, por exemplo, o tamanho médio da fila e o tempo de utilização de um determinado processador em uma mesma execução da simulação.

5.3.1 ASDA-MRIP

A primeira estrutura proposta para o novo ambiente de simulação MRIP pode ser observada na figura 5.7. Segundo este esquema pode-se verificar que existe um processo mestre, o qual é responsável por iniciar os escravos (utilizando o PVM), e vários processos escravos sendo executados em cada máquina pertencente à máquina virtual, responsáveis por iniciar o programa de simulação que o mestre enviou (utilizando *sockets*). Essa estrutura é semelhante à adotada pelo Akaroa (figura 3.17), porém com comunicação somente via *sockets*.

O objetivo em fazer com que o processo escravo iniciasse o programa de simulação seria o de eliminar do programa de simulação todos os detalhes referentes à comunicação com o mestre. No entanto, havia a necessidade de alguma forma de comunicação entre o processo escravo e o programa de simulação, o que acarretaria em mais uma forma de comunicação, além do fato de que esses processos escravos estariam residentes em memória e esta é importante na abordagem MRIP pois os processos possuem granulação grossa.

Sendo assim, em uma segunda versão da estrutura, o programa de simulação foi definido como sendo um processo escravo. Esta nova estrutura permite que se tenha uma quantidade menor de comunicação e permite que o programa escravo não fique residente na memória. O único processo que deve ficar residente em memória é o arquivo *daemon* do pvm (*pvm*). Esta nova estrutura pode ser observada na figura 5.8.

Figura 5.7. Primeira estrutura idealizada para um ambiente de simulação MRIP

Figura 5.8. Estrutura do ambiente para simulação MRIP

A estrutura da figura 5.8 é uma visão mais ampla de todo o processo MRIP. O processo mestre é responsável por iniciar um determinado número de replicações nas máquinas que fazem parte da máquina virtual estabelecida pelo usuário. Cada replicação executa o mesmo programa de simulação, porém a semente de geração dos números aleatórios é diferente em cada uma das replicações. De acordo com o método de análise de saída utilizado, a cada intervalo de tempo, definido aqui como *checkpoint*, as replicações enviam para o mestre as estimativas da variável analisada. O mestre recebe estas estimativas e verifica se a precisão definida pelo usuário foi atingida. Caso isso aconteça, a simulação é encerrada. Caso a precisão ainda não tenha sido atingida, todas as replicações continuam sendo executadas até o

próximo valor de *checkpoint*.

A figura 5.9 ilustra uma expansão da visão apresentada na figura 5.8. Nesta visão, o ASDA-MRIP é composto pelos submódulos *mestre*, *escravo*, *comunicação*, *analizador global*, *analizador local* e *debug*. As funções dos submódulos *mestre* e *escravo*, bem como dos submódulos ligados a eles, já foram detalhadas no parágrafo anterior. O submódulo *comunicação* é o responsável por prover as funções de comunicação utilizando o PVM. Ele é utilizado pelos submódulos *mestre*, *escravo* e *debug*. O submódulo *debug* fornece funções para que o usuário possa acompanhar todo o processo de simulação através de arquivos *logs*.

Figura 5.9. Estrutura detalhada da ASDA-MRIP

A seguir, cada um desses submódulos será detalhado.

5.3.2 Submódulo Mestre

O submódulo *mestre* é responsável por controlar o processo de recebimento das estimativas dos escravos e encaminhá-los ao analisador global. De forma a viabilizar esse controle, o submódulo *mestre* possui duas fases: *pré-ciclo* e *ciclo*.

Na fase **pré-ciclo** tem-se a inicialização das variáveis que armazenam os nomes das máquinas onde os escravos estão sendo executados, a geração das sementes que irão ser enviadas para cada escravo (replicação) e finalmente a inicialização dos escravos.

Na fase **ciclo** o *mestre* permanece em um laço infinito aguardando o envio das estimativas calculadas pelos escravos. Tão logo as observações chegam, estas são

encaminhadas para o analisador global. Se o analisador global chegou à conclusão de que a simulação pode ser encerrada então é enviada uma mensagem para todas as replicações (M_STOP) e os processo são encerrados. Caso contrário, uma mensagem de continuação (M_CONT) é retornada às replicações e estas continuam a simulação até o próximo *checkpoint*, onde todo o processo se repete. A mensagem de continuação da execução da simulação deve ser enviada aos escravos pois, assim que um escravo envia suas estimativas para o mestre, o processo fica aguardando a resposta para que a simulação prossiga ou não.

A geração das sementes que são enviadas para as replicações é realizada de forma sequencial, na fase de pré-ciclo. São geradas tantas sementes quando o número de replicações e estas são armazenadas em um vetor. Assim que o vetor é gerado, uma função percorre-o de forma a verificar se não existem sementes repetidas. Caso exista, uma nova semente é gerada para a primeira posição repetida e o processo de verificação inicia novamente.

5.3.3 Submódulo Analisador Global

Este submódulo tem por função verificar se a precisão requerida pelo usuário foi atingida. Este submódulo é composto basicamente pela função `SimCalcGlobalEstimate()`, a qual é chamada dentro da fase ciclo do submódulo Mestre.

Esta função calcula a média ponderada das estimativas enviadas pelas replicações levando em consideração a quantidade de observações coletadas em cada replicação. Já a função `SimSimulationOver()` verifica se o intervalo de confiança obtido com a média ponderada é suficiente para atingir a precisão definida pelo usuário na variável *precisão*.

Caso a precisão foi atingida, a variável global `simulation_over` torna-se verdadeira e a função `SimCalcGlobalEstimate()` retorna verdadeiro. Caso contrário, é retornado falso.

5.3.4 Submódulo Escravo

O submódulo escravo é encarregado de receber a observação obtida no programa de simulação e direcioná-la ao método de análise selecionado pelo usuário (média dos lotes, análise espectral ou padronização de séries temporais).

Em cada programa de simulação replicado são executadas duas fases (figura 4.21): a de detecção do período transiente e a de estimação dos parâmetros (Pawlikowski, 1990). A seguir, a implementação de cada uma dessas fases é detalhada.

5.3.4.1 Período transiente

Para o estágio de detecção do período transiente deve-se estabelecer um mecanismo que determine quando o período transiente foi ultrapassado. O algoritmo utilizado no ASDA-MRIP é o descrito em (Pawlikowski, 1990), que por sua vez é baseado nos testes propostos por Schruben *et al.* (1983).

Como descrito na seção 2.4.4, várias regras heurísticas (também descritas em (Pawlikowski, 1990)) podem ser utilizadas para uma primeira aproximação do número de observações a serem eliminadas no período transiente. A regra heurística utilizada nesta implementação é a nº 5, descrita a seguir.

Heurística 5: o período inicial transiente foi alcançado após n_0 observações se a série $x_1, x_2, \dots, x_{n_0}$, cruzar a média $\bar{X}(n_0)$ k vezes.

Esta heurística é sensível ao valor da variável k . Valores grandes de k podem conduzir a uma superestimação do valor n_0 e valores pequenos de k podem resultar num valor subestimado de observações a serem descartadas. O valor padrão da variável adotado no presente trabalho é 25, o mesmo adotado no Akaroa.

Um exemplo real desta heurística pode ser observado na figura 5.10. Neste exemplo, o valor de k é 11 e a fase heurística do estágio de determinação do período transiente finaliza na observação número 79. A linha reta representa a média das últimas n_0 (79) observações e a linha que está variando representa as observações coletadas. Pode-se observar que a média foi interceptada pelas observações 11 vezes, o que indica que o período transiente já terminou.

Figura 5.10. Exemplo da heurística no. 5 para detecção do período transiente

Determinada essa primeira aproximação deve-se continuar coletando mais observações para que seja verificado quando a sequência se torna estacionária. Essas observações são utilizadas para o cálculo do estimador da variância da média amostral ($\hat{\sigma}^2[\bar{X}(n)]$) e do grau de liberdade (k). Para esses cálculos podem ser utilizados diversos métodos, sendo que o utilizado no ASDA-MRIP é o baseado na análise espectral das observações e descrito por Heidelberg & Welch (1981).

5.3.4.2 Período de cálculo das estimativas

Após a eliminação das observações coletadas no período transiente inicia-se o estágio

de estimação dos parâmetros. Neste estágio as observações são testadas com o objetivo de verificar quando os resultados atingem o nível de confiança requerido pelo usuário. Conforme descrito na seção 2.4.6, os métodos média dos lotes, padronização de séries temporais e análise espectral podem ser utilizados com esse objetivo. Como cada replicação é um programa de simulação seqüencial, um desses métodos citados deve estar sendo utilizado nas replicações.

O método média dos lotes possui duas fases: a de determinação do tamanho do lote e a de estimação. Deve-se garantir que as médias coletadas em cada replicação não sejam correlacionadas entre si devido ao fato que uma replicação pode, por exemplo, estar sendo executada em uma máquina mais rápida que as outras e enviar duas estimativas seguidas para o mestre. Se essas estimativas são correlacionadas elas podem não fornecer um resultado confiável. Determinado o tamanho do lote, inicia-se a fase de coleta das estimativas que serão enviadas ao mestre. As fórmulas utilizadas nos cálculos dessas duas fases foram descritos em 2.4.5 (correlação) e 2.4.6.2 (método da média dos lotes) .

O método padronização de séries temporais utiliza uma outra abordagem para determinar quando as estimativas podem começar a ser enviadas ao mestre. Como descrito em 2.4.6.3, o método utiliza a fórmula 2.21 como estimador da variância da seqüência. Porém essa fórmula pode ser utilizada somente se os valores A_i obedecerem a uma distribuição normal. Portanto, neste método, as variáveis A_i são calculadas e um teste de normalidade é aplicado sobre elas com o objetivo de determinar quando a fórmula 2.21 pode ser aplicada para o cálculo da variância e, conseqüentemente, do intervalo de confiança.

No método análise espectral não é necessário nenhuma fase para determinação de quando as estimativas são enviadas ao mestre. No início da fase de estimação é calculado um ponto (checkpoint) onde as estimativas serão enviadas ao mestre. O cálculo desse ponto pode ser feito de forma linear ou geométrica, a qual fica a critério do usuário.

Os métodos de análise de saída implementados no ASDA-MRIP são os definidos nesta seção e cuja teoria pode ser encontrada na seção 2.4. No entanto, verificou-se alguns problemas no método média dos lotes, pois este necessita de um período para determinação do tamanho do lote, o que não é muito adequado para a abordagem MRIP. Dessa forma, os métodos Análise Espectral e Padronização de Séries Temporais são os utilizados nos testes que serão descritos no capítulo 6, apesar do método Média dos Lotes também ter sido implementado.

5.3.5 Submódulo Comunicação

O submódulo Comunicação é responsável por todas as rotinas que envolvem a comunicação entre mestre e escravo, sendo que estas podem ser utilizadas tanto pelo mestre quanto pelo escravo. Todas as rotinas estão definidas em um único arquivo, denominado *communication.c*. As rotinas deste arquivo utilizam o PVM como ambiente de passagem de

mensagem e estão listadas na tabela 5.1.

Tabela 5.1. Relação das funções que fazem parte do arquivo *communication.c*

Função	Descrição
<code>int start_communication(void)</code>	Inicia a comunicação com o mestre
<code>Int GetParent(void)</code>	Retorna o identificador do processo pai (mestre)
<code>Int GetID(void)</code>	Retorna o identificador do próprio processo
<code>Int SendCP(int slave, struct Checkpoint cp)</code>	O escravo <code>slave</code> envia a estrutura <code>Checkpoint</code> para o processo mestre
<code>int SendDouble(int who, double value)</code>	Envia para o processo que possui <code>id who</code> uma variável <code>double</code>
<code>int SendLong(int who, long value)</code>	Envia para o processo que possui <code>id who</code> uma variável <code>long</code>
<code>int SendInt(int who, int value)</code>	Envia para o processo que possui <code>id who</code> uma variável inteira
<code>int ReceiveCP(int *slave, struct Checkpoint *cp)</code>	Processo mestre recebe uma estrutura <code>Checkpoint</code>
<code>int ReceiveDouble(int who, double *value)</code>	Recebe do processo que possui <code>id who</code> uma variável <code>double</code>
<code>int ReceiveLong(int who, long *value)</code>	Recebe do processo que possui <code>id who</code> uma variável <code>long</code>
<code>int ReceiveInt(int who, int *value)</code>	Recebe do processo que possui <code>id who</code> uma variável inteira
<code>void exit_PVM(void)</code>	Finaliza o PVM

A rotina mais importante do submódulo comunicação é a *start_communication()* (figura 5.11). Esta função deve estar sempre no início do programa de simulação (processo escravo) e tem como função:

1. receber a primeira mensagem enviada pelo mestre;
2. verificar em que posição da variável `tids` está o identificador do próprio processo;
3. enviar uma nova mensagem de confirmação para o mestre contendo o nome da máquina e o seu identificador; e
4. retornar para o processo que a chamou um código para o método de análise escolhido ou o valor -1 em caso de erro.

Uma ilustração do processo de comunicação entre mestre e escravo pode ser observado na figura 5.12. Em ① o processo mestre inicia a execução dos escravos através da função `pvm_spawn()`. Em ② o processo mestre inicia a comunicação com os escravos. Em ③ os escravos enviam confirmação ao mestre.

```

int start_communication(void)
{
    DUMP_TEXT("Funcao: start_communication()");

    /* enroll in pvm */
    mytid = pvm_mytid();
    DUMP_VALUE("TID:", (real)mytid);

    /* Receive data from master */
    msgtag = 0;
    pvm_rcv( -1, msgtag );
    pvm_upkint(&nproc, 1, 1);
    pvm_upkint(&metodo_analise, 1, 1);
    pvm_upkint(tids, nproc, 1);
    } 1

    /* Determine which slave I am (0 -- nproc-1) */
    for( i=0; i<nproc ; i++ )
        if( mytid == tids[i] )
        {
            me = i;
            break;
        }
    DUMP_VALUE("me:", (real)me);
    } 2

    // Send back to master the host name and the identifier
    if (uname(&host_local)<0)
    {
        DUMP_TEXT("Erro no uname");
        return -1;
    }
    strcpy(hostname, host_local.nodename);
    pvm_initsend(PvmDataDefault);
    pvm_pkint(&mytid, 1, 1);
    pvm_pkstr(hostname);
    pvm_send(pvm_parent(), 0);
    DUMP_TEXT("Fim da start_communication()");
    Return metodo_analise;
    } 3
}

```

Figura 5.11. Código da função *start_communication*

As outras funções do submódulo *comunicação* são funções que realizam o envio e recebimento de dados definidos (*int*, *long*, *double*, etc.) e também da estrutura CP (*checkpoint*), sendo que estão presentes em todos os submódulos.

5.3.6 Submódulo *Debug*

Devido ao fato dos processos escravos não terem como reportar ao usuário o andamento da simulação de uma maneira gráfica (utilizando o dispositivo de saída padrão e em tempo de execução) pois estão sendo executados remotamente, foram implementadas algumas funções que armazenam todos os resultados em arquivos.

Figura 5.12. Comunicação entre o processo mestre e os escravos

Para o processo mestre é criado um arquivo denominado *debugmasterID.log*, onde *ID* é o número do processo mestre. Neste arquivo são armazenados:

- informações sobre o processo a ser executado. Exemplo:

```
Nome do programa: /home/sarita/replicador/final/exemplo
Numero de replicacoes: 9
```

- possíveis mensagens de erro na comunicação com os escravos. Exemplo: mensagem exibida quando a função *pvm_spawn* retorna erro:

```
Trouble spawning slaves. Aborting.
```

- uma lista contendo os identificadores dos escravos e em que máquina estão sendo executados. Exemplo:

identificador do processo	máquina onde o processo está será executado
262150	lasdpc06.icmc.sc.usp.br
786435	lasdpc07.icmc.sc.usp.br
1048579	lasdpc04.icmc.sc.usp.br
1835011	lasdpc08.icmc.sc.usp.br
1310723	lasdpc10.icmc.sc.usp.br
524291	lasdpc05.icmc.sc.usp.br
1572867	lasdpc03.icmc.sc.usp.br
2359299	lasdix.icmc.sc.usp.br
2097155	lasdpc02

- a semente dos números aleatórios que foi enviada para cada escravo. Exemplo:

```
Semente: 100
semente[0] = 90    // semente escravo 0
semente[1] = 41    // semente escravo 1
semente[2] = 87    // semente escravo 2
semente[3] = 30    // semente escravo 3
semente[4] = 3     // semente escravo 4
semente[5] = 61    // semente escravo 5
semente[6] = 11    // semente escravo 6
semente[7] = 51    // semente escravo 7
semente[8] = 9     // semente escravo 8
```

- número do escravo que enviou um *checkpoint*. Exemplo:

```
Recebeu cp de: 786435
```

- valores analisados pela função *SimCalcGlobalEstimate()*. Exemplo:

```
Delivering checkpoint with count 14388, mean: 0.599298, variance:
0.001403, from slave 786435
Global estimate: mean=0.599298 var=0.00140328 delta=0.079432 df=16
g.cp.delta: 0.079432
fabs(g.cp.mean): 0.599298
g.precision: 0.050000
fabs(g.cp.mean)*g.precision: 0.0299649
Replying 2 to slave: 786435
Aguardando mensagens dos escravos
```

- o tempo de execução da simulação. Exemplo:

```
Tempo em seg (difftime): 5.000000
Tempo de execucao (uSegundos) (gettimeofday): 5171241
Tempo de execucao ( Segundos) (gettimeofday): 5.171241
```

Para cada processo escravo, dois arquivos são criados: o *dataID.log* e o *debugID.log*, onde, novamente, *ID* é o identificador de cada processo escravo. O arquivo *dataID.log* armazena os dados gerados pela simulação da variável estimada e o arquivo *debugID.log* armazena informações referentes à replicação, tais como o tamanho do período transiente e os valores de algumas variáveis utilizadas nas decisões (períodos transiente e estimação).

O resultado final da simulação é apresentado na tela da seguinte maneira:

```
***** RESULTADOS FINAIS *****
Estimate      Delta  Conf      Var      Count      Trans
0.598102      0.0289356  0.95  0.000209699  87387      1799
```

onde:

Estimate: é o valor médio da variável analisada

Delta: é metade do intervalo de confiança

Conf: é o nível de confiança

Var: é a variância da variável analisada

Count: é o número total de observações coletadas

Trans: é o número de observações coletadas no período transiente

5.3.7 Estrutura de subdiretórios e execução da simulação

Uma estrutura de subdiretórios foi criada com o objetivo de separar os arquivos que são utilizados somente pelo processo mestre dos que são utilizados somente pelos processos escravos (figura 5.13). Esta estrutura facilita o gerenciamento dos arquivos e possui a seguinte divisão: *mestre*, *escravo*, *comum* e *final*.

Figura 5.13. Estrutura de subdiretórios do ASDA-MRIP

O subdiretório *mestre* possui os arquivos utilizados somente para a construção do arquivo mestre denominado *master*. O subdiretório *escravo* possui o programa de simulação que será executado e também os subdiretórios *STS* (método Padronização de Séries Temporais), *SA* (método Análise Espectral) e *BM* (método Média dos Lotes), sendo que cada um deles implementa, respectivamente, os métodos padronização de séries temporais, análise espectral e média dos lotes. O subdiretório *comum* possui arquivos que são utilizados tanto

pelos processos mestre como pelos processos escravos (comunicação, debug e definição das variáveis). Finalmente, o subdiretório *final* possui os arquivos objeto (.o no linux) de todos os subdiretórios citados acima para que os arquivos executáveis *master* e *nome_arq_simulacao* possam ser gerados. A tabela 5.2 lista os arquivos pertencentes a cada subdiretório e sua respectiva função.

Tabela 5.2. Relação dos arquivos do ASDA-MRIP e suas funções

Localização/nome do arquivo	Descrição
/comum/communication.c (.h)	Rotinas de comunicação
/comum/variables.c	Variáveis globais do ambiente
/comum/debug/c (.h)	Rotinas para acompanhamento dos resultados da simulação
/comum/cmr_random_generator.c (.h)	Geração de números aleatórios pelo método CMR (<i>Combined Multiple Recursive</i>)
/comum/random.c (.h)	Implementação das distribuições de probabilidade
/comum/distributions.h	Nome das distribuições de probabilidade
/comum/normal_distribution.c (.h)	Calcula o valor do p-ésimo quantile de uma distribuição normal
/comum/t_distribution.c (.h)	Calcula o valor do p-ésimo quantile de uma distribuição t-Student
/comum/checkpoint.h	Definição da estrutura <i>Checkpoint</i>
/comum/real.h	Definição da variável <i>real</i> (<i>double</i>)
/comum/boolean.h	Definição do tipo <i>boolean</i> e das variáveis <i>true</i> e <i>false</i>
/mestre/master.c (.h)	Inicia os escravo, recebe as observações enviadas pelos escravos e encerra a simulação
/mestre/checkpoint_analyser.c (.h)	Verifica quando a simulação pode ser encerrada
/mestre/chkpt_queue.c (.h)	Funções para manipulação da pilha da estrutura <i>checkpoint</i>
/escravo/STS/sts_csum.c (.h)	Implementação do método Padronização de Séries Temporais
/escravo/BM/bm_analyser.c (.h)	Implementação do método Média dos Lotes
/escravo/SA/spectral_parameter_analyser.c (.h)	Implementação do método Análise Espectral
/escravo/SA/transient.c (.h)	Deteção do período transiente seguindo o método Análise Espectral
/escravo/SA/steady.c (.h)	Cálculo do período de estimação seguindo o método Análise Espectral
/escravo/SA/variance.c (.h)	Estimativa da variância seguindo o método Análise Espectral
/escravo/SA/leastsquare.c (.h)	Ajuste de uma função polinomial pelo método dos mínimos quadrados

Para executar uma simulação deve-se digitar na linha de comando no subdiretório *final* a seguinte seqüência:

master <prog_simulacao> <num_replicacoes> <semente> <met_analise>

onde:

master: processo mestre

<prog_simulacao>: executável do programa de simulação a ser replicado

<num_replicacoes>: número de replicações

<semente>: semente para o gerador de números aleatórios que vai gerar as sementes de cada replicação

<met_analise>: método de análise a ser utilizado

Para o último parâmetro, os possíveis valores são: *sts*: padronização de séries temporais, *sa*: análise espectral e *bm*: média dos lotes.

A estrutura de um programa de simulação para ser executado utilizando o ASDA-MRIP pode ser observado na próxima seção, onde será detalhada a implementação do módulo Interface de Software.

5.4 Módulo Interface de Software

O Módulo Interface de Software é responsável por fornecer ao usuário a possibilidade de utilização de programas de simulação seqüenciais ou que utilizem a abordagem SRIP já desenvolvidos. Para isso, deve-se introduzir no código do programa de simulação algumas linhas necessárias para tornar o programa de simulação um programa de simulação distribuída que utiliza a abordagem MRIP. Para que isso seja possível, o usuário deve fornecer o nome do arquivo que deseja replicar e também em que linha do código está a definição da variável que vai ser observada.

É importante ressaltar que o que será descrito a seguir é válido para programas de simulação que utilizam a linguagem C como linguagem hospedeira pois é nesta linguagem que o ASDA-MRIP foi desenvolvido.

Para a implementação deste módulo, definiu-se um conjunto de trechos de código que devem ser inseridos nos programas de simulação que o usuário deseja replicar. Foram definidos oito trechos que devem ser inseridos no programa de simulação:

1. Definição dos arquivos **#include**;
2. Definição das variáveis;
3. Definição e inicialização dos arquivos de acompanhamento da simulação;
4. Início da comunicação;

5. Inicialização do método de análise;
6. Recebimento da semente gerada pelo mestre e inicialização do gerador de números aleatórios;
7. Coleta da estimativa a ser observada
8. Finalização dos arquivos de acompanhamento da simulação

A figura 5.14 ilustra os oito trechos.

1	<pre>... #include "distributions.h" #include "debug.h" #include "communication.h" ... int metodo_analise; int id; int parent; int seed; ... id = GetID(); parent = GetParent(); if (Open_Debug_File(id) == 0) { DUMP_TEXT("Erro na abertura do arquivo de debug"); Close_Debug_File(); SendInt(parent,M_EAE); exit(0); } if (Open_Data_File(id) == 0) { DUMP_TEXT("Erro na abertura do arquivo de dados"); Close_Data_File(); Close_Debug_File(); SendInt(parent,M_EAE); exit(0); } ... metodo_analise = start_communication(); if (metodo_analise == -1) { DUMP_TEXT("Erro no inicio da comunicacao com o mestre"); Close_Debug_File(); Close_Data_File(); SendInt(parent,M_ECE); Exit(0); } ... Inicializa_MetodoAnalise(metodo_analise); ... ReceiveInt(parent,&seed); Inicializa_CMRRandomGenerator((unsigned long)seed); ... // Logo após o cálculo da estimativa a ser observada ProcessObservation(estimativa); ... // Antes do programa finalizar Close_Degub_File(); Close_Data_File();</pre>
---	--

Figura 5.14. Trechos a serem inseridos no código do programa de simulação

O trecho de definição dos arquivos **#include** insere no código as linhas **#include “distributions.h”**, **#include “communication.h”** e **#include “debug.h”**. Esses arquivos contêm as definições das funções de comunicação (*communication.h*) e acompanhamento da simulação (*debug.h*).

O trecho de definição das variáveis insere no código as variáveis necessárias para que a abordagem MRIP possa ser implementada: *id* e *parent*. A variável *metodo_analise* armazena o código do método de análise escolhido pelo usuário, valor retornado pela função *start_communication()*; a variável *id* é utilizada para armazenar o identificador do processo.

O trecho de definição e inicialização dos arquivos de acompanhamento da simulação deve primeiramente obter os números dos identificadores do próprio processo e do processo pai, armazenando-os, respectivamente, nas variáveis *id* e *master*. Logo em seguida, as funções *Open_Debug_File(id)* e *Open_Data_File(id)* abrem, em modo *append*, os arquivos de acompanhamento *debugID.log* e *dataID.log*, onde *ID* é o valor da variável *id*. Caso ocorra erros na criação/abertura dos arquivos, uma mensagem de erro é escrita no arquivo de acompanhamento da simulação, os arquivos abertos são fechados, é enviada uma mensagem ao processo pai com o código M_EAE (erro em arquivos do escravo) e o processo é finalizado.

Para o trecho de início da comunicação é necessária a inserção da função *start_communication()*. Caso o valor retornado por esta função seja -1, isto significa que o início da comunicação falhou. Neste caso, os arquivos de acompanhamento são fechados, é enviada uma mensagem ao processo pai com o código M_ECE (erro na comunicação do escravo) e o processo é finalizado.

O trecho de inicialização do método de análise utiliza o valor retornado pela função *start_communication()* (o qual é a opção do método de análise escolhido pelo usuário e enviado pelo mestre) e o utiliza como parâmetro da função *Inicializa_MetodoAnalise()*. Esta função define as variáveis globais do método escolhido.

O trecho de recebimento da semente gerada pelo mestre e inicialização do gerador de números aleatórios utiliza a função *ReceiveInt()* para receber do processo mestre o valor da semente e utiliza também a função *Inicializa_CMRRandomGenerator()* para definir a semente do gerador de números aleatórios.

O último trecho de código que deve ser inserido dentro do laço da simulação e logo após o cálculo da variável a ser observada é composto pela função *ProcessObservation()*. Esta função envia a estimativa para o método de análise escolhido e este realiza todo o processo necessário para determinar se os valores já podem ser enviados para o mestre ou não.

Para finalizar, os arquivos de acompanhamento da simulação devem ser fechados através das funções *Close_Debug_File()* e *Close_Data_File()*.

Todos esses passos já foram definidos para inserção em 3 bibliotecas de simulação: SMPLX (Uilson, 1999); Simpack, versão orientada a objetos da extensão SMPL (Fishwick, 1992) e SSS, *Simulation Subroutine Set* (Pollatschek, 1997). Os programas gerados podem ser observados no Apêndice C, nas figuras C.1, C.2 e C.3, respectivamente.

Como pode ser observado, os programas escritos utilizando as bibliotecas SMPL e SSS são parecidos pois não utilizam o conceito de orientação a objetos. Já o programa que utiliza a biblioteca SimPack possui uma formatação diferente.

Além da inserção dos trechos descritos, existe também a necessidade de substituir no programa de simulação as funções de estatística definidas no código original pelas funções definidas no arquivo “*distribution.c*”. A tabela 5.3 ilustra a relação entre as funções de estatística definidas nas 3 bibliotecas (SMPL, SimPack e SSS) e a utilizada pelo ASDA-MRIP.

Tabela 5.3. Relação das funções de funções de probabilidade nas bibliotecas ASDA-MRIP, SMPLX, SimPack e SSS

ASDA-MRIP	SMPLX	SimPack	SSS
real ranf()	int randompar(int i, int n)	int random(int i, int n)	double RA(void)
real Uniform(real a, real b)	double uniform(double a, double b)	double uniform(double a, double b)	double UN(double I,double C)
long UniformInt(long m, long n)	-	-	-
real Exponential(real x)	double expntl(double x)	double expntl(double x)	double EX(double M)
Real Erlang(real x, real s)	double erlang(double x, double s)	double erlang(double x, double s)	double ER(double M,int K)
real HyperExponential(real x, real s)	double hyperx(double x, double s)	double hyperx(double x, double s)	-
real Normal(real x, real s)	double normal(double x, double s)	double normal(double x, double s)	double RN(double M,double S)
real LogNormal(real x, real s)	-	-	double RL(double M,double S)
long Geometric0(real x)	-	-	-
real HyperGeometric(real x, real s)	-	-	-
long Binomial(long n, real u)	-	-	int BI(int N,double P)
long Poisson(real x)	-	-	int NP(double M)
-	-	-	double GA(double M,double K)
-	-	-	double BE(double U,double W)
-	-	double triang (double a, double c, double b)	double TR(double I,double B,double C)
real Weibull(real alpha, real beta)	-	-	double WE(double M,double K)

5.5 Considerações Finais

Este capítulo apresentou a implementação de três módulos do ambiente proposto nesta tese: Módulo Interface com o Usuário, Módulo Replicador e Módulo de Interface de Software.

A escolha pela implementação desses módulos deve-se a vários fatores, dentre eles: a necessidade de fornecer uma interface para o usuário interagir com o sistema e em segundo lugar porque, dentro das pesquisas desenvolvidas dentro do grupo de Sistemas Distribuídos e Programação Concorrente do ICMC – USP, a abordagem MRIP ainda não havia sido pesquisada. Desta forma, a prioridade no desenvolvimento de um protótipo do ASDA foi para os módulos Interface com o Usuário e Replicador.

Para a implementação do módulo Interface com o Usuário optou-se pela utilização do software Kylix, da Borland. Faz parte da implementação deste módulo a definição da base de dados a partir das classes definidas na especificação do ASDA (capítulo 4) e o desenvolvimento da interação com o usuário.

O módulo Replicador foi implementado tendo como base a estrutura do software Akaroa. A comunicação entre os processos foi desenvolvida para utilização do ambiente de passagem de mensagem PVM.

Como o módulo Interface de Software está diretamente ligado ao módulo Replicador, este também foi implementado com a definição dos trechos de código necessários para tornar um programa de simulação sequencial em um programa de simulação distribuída utilizando a abordagem MRIP. Foi definido como esses trechos são inseridos em programas de simulação já desenvolvidos utilizando as bibliotecas de simulação sequencial: SMPLX (versão da biblioteca SMPL para linux), SimPack (versão orientada a objetos da biblioteca SMPL) e SSS.

O próximo capítulo utiliza dois modelos de sistemas computacionais de modo a validar a implementação do módulo Replicador.

Testes realizados com o ASDA-MRIP

6.1 Considerações Iniciais

Dentre as opções para o desenvolvimento de simulação distribuída, a abordagem MRIP ainda não havia sido trabalhada dentro do grupo de Sistemas Distribuídos e Programação Concorrente do ICMC – USP. Como descrito no capítulo 5, este foi um dos motivos para a escolha do desenvolvimento do módulo Replicador, o ASDA-MRIP.

A princípio, os testes do módulo replicador foram projetados com o objetivo de se validar a implementação do ASDA-MRIP desenvolvida neste trabalho de doutorado. Os resultados foram validados em duas etapas: uma primeira valida o resultado final segundo o nível de confiança e precisão estabelecidos pelo usuário, e uma segunda valida o resultado final da simulação através da comparação com soluções analíticas e simulações utilizando outros softwares. Em todos os testes o programa de simulação utilizado foi desenvolvido na linguagem C com a extensão funcional SMPLX (Ulson, 1999). Dois exemplos dos resultados utilizados na validação são apresentados neste capítulo através dos modelos de um sistema M/M/1 e de um sistema distribuído.

Uma vez concluída a validação do módulo Replicador, os testes realizados foram ampliados com o intuito de se realizar a avaliação de algumas características da implementação MRIP desenvolvida neste trabalho.

Todos os testes foram realizados no laboratório do grupo de Sistemas Distribuídos e Programação Concorrente do ICMC – USP. As seções 6.2 e 6.3 descrevem a metodologia utilizada durante a realização dos testes e as características da arquitetura computacional onde estes foram realizados. A seção 6.4 descreve a validação dos resultados e a 6.5 a análise dos resultados. O Apêndice C possui o código dos programas utilizados.

6.2 Metodologia adotada

A obtenção dos resultados para validação do ASDA-MRIP e para análise da abordagem MRIP para simulação distribuída baseou-se em diversas execuções com a variação de diversos parâmetros. Em todos os testes foram ativadas diversas replicações, utilizando-se diferentes métodos de análise e níveis de confiança. Para tornar os resultados representativos, execuções com diferentes sementes foram utilizadas. A ativação da simulação é realizada pelo comando:

```
master <prog_simulacao> <num_replicacoes> <semente> <met_analise>
```

onde:

prog_simulacao: nome do programa de simulação a ser replicado

num_replicacoes: número de replicações a serem realizadas

semente: semente principal que gerará as sementes de cada replicação

met_analise: método de análise a ser utilizado

Além da variação dos parâmetros da linha de comando, foi variado também o nível de confiança: 0,95 (95%) e 0,99 (99%), o qual é definido no arquivo `/comum/variables.c`. O valor da precisão manteve-se o mesmo em todos os testes: 0,05 (5%).

Nos testes realizados para a validação do ASDA-MRIP, os parâmetros receberam os seguintes valores:

num_replicacoes: 9, 18

semente: 100, 500, 6543 e 9765

método de análise: sa (análise espectral)

sts (padronização de séries temporais)

A combinação desses parâmetros resultou em um total de 32 execuções para cada um dos dois modelos utilizados, como pode ser observado na tabela 6.1.

Tabela 6.1. Configuração dos testes realizados para os modelo de validação

Execução	Nº replicações	Semente	Método Análise	Nível de confiança
1	9	100	sa	0.95
2	9	500	sa	0.95
3	9	6543	sa	0.95
4	9	9765	sa	0.95
5	9	100	sts	0.95
6	9	500	sts	0.95
7	9	6543	sts	0.95
8	9	9765	sts	0.95
9	18	100	sa	0.95
10	18	500	sa	0.95
11	18	6543	sa	0.95
12	18	9765	sa	0.95
13	18	100	sts	0.95
14	18	500	sts	0.95
15	18	6543	sts	0.95
16	18	9765	sts	0.95

Execução	Nº replicações	Semente	Método Análise	Nível de confiança
17	9	100	sa	0.99
18	9	500	sa	0.99
19	9	6543	sa	0.99
20	9	9765	sa	0.99
21	9	100	sts	0.99
22	9	500	sts	0.99
23	9	6543	sts	0.99
24	9	9765	sts	0.99
25	18	100	sa	0.99
26	18	500	sa	0.99
27	18	6543	sa	0.99
28	18	9765	sa	0.99
29	18	100	sts	0.99
30	18	500	sts	0.99
31	18	6543	sts	0.99
32	18	9765	sts	0.99

Nos testes realizados com o objetivo de analisar o comportamento da abordagem MRIP, os parâmetros receberam os seguintes valores:

num_replicacoes: 12, 24 e 36

semente: 100, 500, 537, 1578, 2198, 3333, 5279, 6543, 8484 e 9765

método de análise: sa (análise espectral)

sts (padronização de séries temporais)

Para todos os testes foi mantido o nível de confiança em 0,95 (95%) e o valor da precisão em 0,05 (5%).

A combinação desses parâmetros resultou em um total de 60 execuções para cada um dos três modelos testados, como pode ser observado na tabela 6.2.

Para a fase de análise houve a necessidade de alterações de três parâmetros em relação ao conjunto de testes realizado na fase de validação. A primeira alteração foi aumentar o número de máquinas e conseqüentemente, o número de replicações que podem ser executadas simultaneamente. A segunda alteração foi aumentar o número de sementes utilizadas. Como o objetivo é fazer a análise de modo que conclusões com bases estatísticas possam ser obtidas, o número de sementes aumentou de 4 para 10, possibilitando assim conclusões estatísticas mais confiáveis. A terceira alteração foi a utilização de um terceiro modelo. Esse terceiro modelo é uma cópia do segundo modelo (sistema distribuído), porém com o acréscimo de uma repetição *while* no corpo do programa. Este modelo será explicado na seção 6.5.

Tabela 6.2. Configuração dos testes realizados para os modelo de análise

Execução	Nº replicações	Semente	Método Análise
1	12	100	sa/sts
2	12	500	sa/sts
3	12	537	sa/sts
4	12	1578	sa/sts
5	12	2198	sa/sts
6	12	3333	sa/sts
7	12	5279	sa/sts
8	12	6543	sa/sts
9	12	8484	sa/sts
10	12	9765	sa/sts
11	24	100	sa/sts
12	24	500	sa/sts
13	24	537	sa/sts
14	24	1578	sa/sts
15	24	2198	sa/sts
16	24	3333	sa/sts
17	24	5279	sa/sts
18	24	6543	sa/sts
19	24	8484	sa/sts
20	24	9765	sa/sts
21	36	100	sa/sts
22	36	500	sa/sts
23	36	537	sa/sts
24	36	1578	sa/sts
25	36	2198	sa/sts
26	36	3333	sa/sts
27	36	5279	sa/sts
28	36	6543	sa/sts
29	36	8484	sa/sts
30	36	9765	sa/sts

Em todos os testes, o valor analisado é coletado através da função *utilizacao_recurso()*, a qual retorna o valor da utilização de um determinado centro de serviço. Para cada replicação, o sistema armazena em arquivos os valores das seguintes métricas:

- Número de observações coletadas no período transiente;
- Número total de observações coletadas;
- Valor médio da variável observada;
- Variância da variável observada;

Como resultado final da simulação, é fornecido ao usuário os seguintes valores:

- Tempo da simulação;
- Valor médio da variável observada;
- Variância da variável observada;
- Delta (metade do intervalo de confiança);
- Número total de observações coletadas;
- Número total de observações coletadas no período transiente;

6.3 Características da arquitetura

Em todas as execuções realizadas foi utilizada a rede de computadores local disponível no Laboratório de Sistemas Distribuídos e Programação Concorrente (LaSDPC) do ICMC – USP São Carlos. A rede de 100 Mbits/seg é interligada através de um hub e composta por 14 computadores pessoais (arquitetura Intel), executando o sistema operacional Linux com as distribuições Conectiva, versão 5.1 e kernel 2.2.16-13cl e Red Hat, versão 7.2, kernel 2.4.7-10. A tabela 6.3 lista a configuração (processador, memória e disco) das 14 máquinas utilizadas. A versão do PVM é a 3.4.2.

Para a execução dos testes a rede local foi isolada de qualquer tráfego externo e também não havia nenhuma outra aplicação sendo executada em cada uma das máquinas.

Tabela 6.3. Configuração das máquinas utilizadas nos experimentos

Máquina	Processador	Memória	Disco
lasdix	Pentium MMX 200 MHz	32 Mb	13 Gb
lasdpc01	Pentium 166 MHz	32 Mb	3 Gb
lasdpc02	Pentium 100 MHz	32 Mb	1,2 Gb
lasdpc03	Pentium MMX 200 MHz	32 Mb	3 Gb
lasdpc04	Pentium II 400 MHz	128 Mb	13 Gb
lasdpc05	Pentium II MMX 266 MHz	256 Mb	6 Gb
lasdpc06	Pentium III 450 MHz	256 Mb	8 Gb
lasdpc07	Pentium II 400 MHz	256 Mb	13 Gb
lasdpc08	Pentium MMX 200 MHz	64 Mb	4 Gb
lasdpc09	Pentium 166 MHz	32 Mb	4 Gb
lasdpc10	Pentium MMX 200 MHz	32 Mb	3 Gb
lasdpc11	Pentium 166 MHz	32 Mb	3 Gb

Máquina	Processador	Memória	Disco
lasdpc12	Pentium II MMX 350 MHz	128 Mb	8 Gb
lasdpc13	Pentium 120 MHz	32 Mb	4 Gb

6.4 Validação do ASDA-MRIP

Os testes descritos na seção 6.2 e utilizados na validação do ASDA-MRIP têm o objetivo de não somente comprovar que os resultados fornecidos pelo ambiente estão de acordo com os resultados esperados como também comprovar que os resultados estão de acordo com o nível de confiança e precisão estabelecidos pelo usuário.

Para comprovar que os resultados fornecidos pelo ASDA-MRIP satisfazem o nível de confiança e a precisão requeridas pelo usuário, foi calculado o valor da variável $t_{n-1,1-\alpha/2}$ da fórmula para o cálculo do valor de delta (metade do intervalo de confiança):

$$\Delta_x = t_{n-1,1-\alpha/2} \hat{\sigma}[\bar{X}(n)] \implies t_{n-1,1-\alpha/2} = \frac{\Delta_x}{\hat{\sigma}[\bar{X}(n)]}$$

Os valores Δ_x e $\hat{\sigma}[\bar{X}(n)]$ são, respectivamente, o delta e a raiz quadrada da variância (desvio padrão) fornecidos pelo ambiente ASDA-MRIP como resultado final.

O valor $t_{n-1,1-\alpha/2}$ é o valor crítico da distribuição t-Student no ponto $1-\alpha/2$ ($1-\alpha$ é o nível de confiança). Para amostras acima de 120 observações, a distribuição t-Student se aproxima dos valores de uma distribuição normal, que é a utilizada nesta tese. Para o nível de confiança 0.95, o valor do ponto crítico é 1.96 e para o nível de confiança 0.99, o valor do ponto crítico é 2.5758 (Meyer, 2000).

Para verificar se a precisão desejada foi atingida basta multiplicar o valor da média fornecido no resultado final da simulação pela precisão desejada. Para que a precisão tenha sido atingida, o valor do delta tem que ser menor do que o valor resultante da multiplicação.

Para comprovar que os resultados fornecidos pelo ASDA-MRIP estão de acordo com os esperados, foi realizada uma comparação entre os resultados obtidos com a simulação dos modelos no ASDA-MRIP e os obtidos através da solução destes mesmos modelos utilizando técnicas analíticas e outros ambientes de simulação.

A seguir são mostrados os resultados e as validações para dois modelos, um sistema M/M/1 e um sistema distribuído, utilizando uma máquina virtual composta por 9 máquinas: lasdpc02, lasdpc03, lasdpc04, lasdpc05, lasdpc06, lasdpc07, lasdpc08, lasdpc10 e lasdix.

6.4.1 Sistema M/M/1

O primeiro modelo testado é um sistema M/M/1, ou seja, um sistema com um servidor e distribuições exponencial para taxa de chegada e taxa de serviço, como pode ser observado na figura 6.1. Este modelo foi resolvido analiticamente em (Silva, 2000) e os resultados fornecidos neste trabalho são utilizados para validar o sistema. O valor da taxa de chegada é 0.0833 e da taxa de serviço é 0.05.

Figura 6.1. Modelo de um sistema M/M/1

Para este modelo, o valor analisado é a utilização do centro de serviço denominado “servidor”. O resultado analítico utilizando o método AVM (Análise do Valor Médio) para modelos abertos e fornecido em (Silva, 2000) é 0.6, ou seja, o centro de serviço tem 60% de utilização.

O código do programa de simulação desenvolvido para a realização deste teste pode ser encontrado na figura C.4 do Apêndice C, e utiliza a biblioteca SMPLX (Ulson, 1999). Os trechos do programa de simulação que estão em preto foram gerados utilizando o programa ASiA (Ambiente de Simulação Automático) (Bruschi, 1999) e os trechos em negrito foram inseridos pelo módulo Interface de Software.

Para a primeira fase da validação, que é a comprovação de que o ambiente fornece resultados de acordo com a precisão e o nível de confiança desejados pelo usuário, a tabela 6.4 compila os resultados da média, da variância, do $t_{n-1, 1-\alpha/2}$, do delta, e do valor resultante da multiplicação da média pela precisão. Nesta tabela, as execuções também associadas à tabela 6.1.

Como pode ser observado, nas execuções 1 a 16 o valor da distribuição t-Student é sempre maior do que 1,96 e nas execuções 17 a 32 o valor da distribuição t-Student também é sempre maior do que 2.5758. Os valores do delta também são sempre menores do que o valor resultante da multiplicação da média pela precisão. Com esses dados, pode-se dizer que os valores fornecidos pelo ASDA-MRIP satisfazem os níveis de confiança e precisão estabelecidos pelo usuário.

Tabela 6.4. Resultados do sistema M/M/1 - validação do nível de confiança e precisão

Execução	Média	Variância	Delta	t-Student	Média * Precisão
1	0,5981	0,0002097	0,028936	1,998179	0,029905
2	0,60042	0,00012096	0,022118	2,011096	0,030021
3	0,61027	0,00019879	0,028726	2,037407	0,0305135
4	0,60251	0,0001697	0,02603	1,998178	0,0301255
5	0,60035	0,00022802	0,029603	1,960394	0,0300175
6	0,60788	0,00023114	0,029804	1,960393	0,030394
7	0,60192	0,00023013	0,029739	1,960391	0,030096
8	0,60534	0,00022703	0,029538	1,960391	0,030267
9	0,59716	0,00021279	0,029148	1,99818	0,029858
10	0,59846	0,00015953	0,025401	2,011094	0,029923
11	0,60551	0,00014558	0,024265	2,011094	0,0302755
12	0,59966	0,00010804	0,021177	2,037409	0,029983
13	0,60038	0,00023047	0,029761	1,960393	0,030019
14	0,6008	0,00021461	0,028719	1,960394	0,03004
15	0,60096	0,00023273	0,029907	1,960395	0,030048
16	0,6017	0,00023364	0,029965	1,960391	0,030085
17	0,59861	0,00010203	0,026657	2,639123	0,0299305
18	0,59714	0,00010851	0,02766	2,655301	0,029857
19	0,60358	0,00011958	0,029036	2,655298	0,030179
20	0,59719	0,00007684	0,023134	2,639125	0,0298595
21	0,60085	0,00013477	0,029908	2,576228	0,0300425
22	0,60691	0,00013789	0,030251	2,576227	0,0303455
23	0,60227	0,00013472	0,029902	2,576231	0,0301135
24	0,60119	0,00009483	0,025088	2,576232	3,01E-02
25	0,60281	0,00011826	0,028583	2,628407	0,0301405
26	0,59759	0,00011672	0,028983	2,682655	0,0298795
27	0,60537	0,000125	0,029687	2,655298	0,0302685
28	0,59936	0,00012539	0,029196	2,60732	0,029968
29	0,60099	0,00013507	0,029941	2,576237	0,0300495
30	0,60265	0,00012451	0,028747	2,576235	0,0301325
31	0,59909	0,00013466	0,029895	2,576225	0,0299545
32	0,60131	0,00010459	0,026347	2,576239	0,0300655

Para a segunda fase da validação, os valores da média e do delta obtidos em cada execução foram compilados na tabela 6.5, sendo que as execuções estão associadas às definidas na tabela 6.1. Como pode ser observado, os valores obtidos nas 32 execuções estão bem próximos do valor calculado pelo método analítico (0,6). Com o objetivo de facilitar a visualização dos resultados e a conseqüente validação do ambiente, o gráfico da figura 6.2 ilustra as 32 execuções (eixo Y) e seus respectivos intervalos de confiança (eixo X). Para cada valor no eixo Y do gráfico, ou seja, para cada execução, existe a representação do intervalo de confiança no eixo X. O intervalo de confiança é composto pelos pontos extremos: (média –

delta, média + delta). A reta traçada no valor $X = 0,6$ representa o valor obtido com a solução analítica. Como pode ser observado, este valor pertence aos intervalos de confiança de todas as execuções.

Tabela 6.5. Resultados obtidos nas execuções do sistema M/M/1

Execução	Média	Delta
1	0,5981	0,0289356
2	0,60042	0,0221184
3	0,61027	0,028726
4	0,60251	0,0260299
5	0,60035	0,0296026
6	0,60788	0,0298041
7	0,60192	0,0297394
8	0,60534	0,0295379
9	0,59716	0,0291483
10	0,59846	0,025401
11	0,60551	0,024265
12	0,59966	0,021177
13	0,60038	0,0297612
14	0,6008	0,028719
15	0,60096	0,0299066
16	0,6017	0,0299654
17	0,59861	0,0266571
18	0,59714	0,0276599
19	0,60358	0,0290359
20	0,59719	0,0231345
21	0,60085	0,029908
22	0,60691	0,0302512
23	0,60227	0,0299016
24	0,60119	0,0250887
25	0,60281	0,028583
26	0,59759	0,0289831
27	0,60537	0,0296869
28	0,59936	0,0291957
29	0,60099	0,0299406
30	0,60265	0,0287469
31	0,59909	0,0298952
32	0,60131	0,026347

Figura 6.2. Intervalo de confiança das execuções do modelo M/M/1

6.4.2 Sistema Computacional Distribuído

O segundo modelo é o descrito por Francês em sua tese de doutorado (Francês, 2001). Este modelo, que pode ser observado na figura 6.3, simula um sistema computacional distribuído e avalia o desempenho do PVM na escolha das máquinas para a execução das tarefas sem a utilização do AMIGO e com a utilização do AMIGO. A única diferença entre essas duas situações é o valor da probabilidade do cliente (neste caso, do processo) ser enviado para uma das 4 máquinas (lasdix, lasdpc08, lasdpc10, lasdpc11) disponíveis. O modelo considerado nos testes foi o sem a utilização do AMIGO, implicando na probabilidade de 25% para cada máquina. Este modelo foi descrito e solucionado utilizando o método analítico AVM (Análise do Valor Médio) e as abordagens redes de filas e redes de Petri, possuindo também dados para que o ASDA-MRIP seja validado. Os valores das taxas de serviço são: centro de serviço PVM = 1,53686, centro de serviço lasdix = 31,32981, centro de serviço lasdpc8 = 82,705, centro de serviço lasdpc10 = 135,7939 e centro de serviço lasdpc11 = 136,6733.

Figura 6.3. Modelo de um sistema computacional distribuído

Para este modelo, o valor analisado é a utilização do centro de serviço denominado “pvm”. Para este centro de serviço, os resultados fornecidos em (Francês, 2001) são:

- Análise do Valor Médio: 0,044831, ou seja, 4,48% de utilização;
- Redes de Filas: $0,0447 \pm 0,00002837$ ou seja, 4,47% de utilização;
- Redes de Petri: $0,0458 \pm 0,0084$, ou seja, 4,58% de utilização;

O código do programa de simulação utilizando o SMPLX desenvolvido para a realização deste teste pode ser encontrado na figura C.5 do Apêndice C. Novamente, os trechos do programa de simulação que estão em preto foram gerados utilizando o programa ASiA (Ambiente de Simulação Automático) e os trechos em negrito foram inseridos pelo módulo Interface de Software.

Da mesma forma que no teste do sistema M/M/1, para a primeira fase da validação, que é a comprovação de que o ambiente fornece resultados de acordo com a precisão e o nível de confiança desejados pelo usuário, a tabela 6.6 compila os resultados da média, da variância, do $t_{n-1, 1-\alpha/2}$, do delta, e do valor resultante da multiplicação da média pela precisão. Nesta tabela, as execuções também associadas à tabela 6.1.

Como pode ser observado, nas execuções 1 a 16 o valor da distribuição t-Student é sempre maior do que 1,96 e nas execuções 17 a 32 o valor da distribuição t-Student também é sempre maior do que 2.5758. Os valores do delta também são sempre menores do que o valor resultante da multiplicação da média pela precisão. Com esses dados, pode-se dizer que os valores fornecidos pelo ASDA-MRIP satisfazem os níveis de confiança e precisão estabelecidos pelo usuário.

Tabela 6.6. Resultados do sistema distribuído - validação do nível de confiança e precisão

Execução	Média	Variância	Delta	t-Student	Média * Precisão
1	0,04522	1,26E-06	0,002225	1,979114	0,002261
2	0,04543	1,19E-06	0,002158	1,979106	0,0022715
3	0,045	1,26E-06	0,002223	1,977008	0,00225
4	0,04516	1,30E-06	0,002251	1,977018	0,002258
5	0,04497	1,27E-06	0,00221	1,960392	0,0022485
6	0,04536	1,34E-06	0,002267	1,960392	0,002268
7	0,04497	1,27E-06	0,00221	1,960392	0,0022485
8	0,04497	1,27E-06	0,00221	1,960392	0,0022485
9	0,04555	1,25E-06	0,002204	1,969704	0,0022775
10	0,04574	1,30E-06	0,002246	1,971863	0,002287
11	0,04547	1,26E-06	0,002208	1,969707	0,0022735
12	0,04545	1,33E-06	0,00227	1,969157	0,0022725
13	0,04497	1,27E-06	0,00221	1,960392	0,0022485
14	0,04536	1,34E-06	0,002267	1,960392	0,002268
15	0,04497	1,27E-06	0,00221	1,960392	0,0022485
16	0,04573	1,35E-06	0,002281	1,96039	0,0022865
17	0,04499	7,25E-07	0,002227	2,615204	0,0022495
18	0,04517	7,19E-07	0,002218	2,615149	0,0022585
19	0,04492	7,33E-07	0,002235	2,610821	0,002246
20	0,04497	7,32E-07	0,002226	2,60147	0,0022485
21	0,04492	7,56E-07	0,00224	2,576234	0,002246
22	0,04522	7,66E-07	0,002255	2,576238	0,002261
23	0,04492	7,56E-07	0,00224	2,576234	0,002246
24	0,04497	7,60E-07	0,002246	2,576236	0,0022485
25	0,04542	7,57E-07	0,002256	2,593422	0,002271
26	0,04544	7,58E-07	0,002261	2,59688	0,002272
27	0,04523	7,55E-07	0,002255	2,595584	0,0022615
28	0,04523	7,35E-07	0,002224	2,594437	0,0022615
29	0,04492	7,56E-07	0,00224	2,576234	0,002246
30	0,04522	7,66E-07	0,002255	2,576238	0,002261
31	0,04492	7,56E-07	0,00224	2,576234	0,002246
32	0,044591	7,47E-07	0,002227	2,576235	0,0022955

Para a segunda fase da validação, os valores da média e do delta obtidos em cada execução foram compilados na tabela 6.7, sendo que as execuções estão associadas às definidas na tabela 6.1. Como pode ser observado, os valores obtidos nas 32 execuções estão bem próximos do valor calculado pelo método analítico (0,0448).

Com o objetivo de facilitar a visualização dos resultados e a conseqüente validação do ambiente, o gráfico da figura 6.4 ilustra as 32 execuções (eixo Y) e seus respectivos intervalos de confiança (eixo X). Para cada valor no eixo Y do gráfico, ou seja, para cada execução,

existe a representação do intervalo de confiança no eixo X. O intervalo de confiança é composto pelos pontos extremos: (média – delta, média + delta). A reta traçada no valor $X = 0,0448$ representa o valor obtido com a solução analítica. Como pode ser observado, este valor pertence aos intervalos de confiança de todas as execuções.

Tabela 6.7. Resultados obtidos nas execuções do sistema distribuído

Execução	Média	Delta
1	0,04522	0,00222532
2	0,04543	0,00215767
3	0,045	0,00222282
4	0,04516	0,00225092
5	0,04497	0,00220998
6	0,04536	0,00226709
7	0,04497	0,00220998
8	0,04497	0,00220998
9	0,04555	0,00220408
10	0,04574	0,00224602
11	0,04547	0,00220805
12	0,04545	0,00226979
13	0,04497	0,00220998
14	0,04536	0,00226709
15	0,04497	0,00220998
16	0,04573	0,0022814
17	0,04499	0,00222676
18	0,04517	0,0022175
19	0,04492	0,00223498
20	0,04497	0,00222574
21	0,04492	0,00224038
22	0,04522	0,0022553
23	0,04492	0,00224038
24	0,04497	0,00224589
25	0,04542	0,00225593
26	0,04544	0,00226142
27	0,04523	0,0022553
28	0,04523	0,00222425
29	0,04492	0,00224038
30	0,04522	0,0022553
31	0,04492	0,00224038
32	0,04498	0,00224621

Figura 6.4. Intervalo de confiança das execuções do modelo sistema distribuído

Com estes testes realizados provou-se que os resultados fornecidos pelo ASDA-MRIP estão de acordo com o nível de confiança e precisão estabelecidos pelo usuário. Com essa certeza, pode-se comprovar que os resultados fornecidos também estão de acordo com os resultados esperados, pois quando comparados com soluções analíticas, estas fazem parte do intervalo de confiança fornecido pelo ambiente.

6.5 Análise dos resultados obtidos

Os testes descritos nas sessões anteriores foram realizados tendo como objetivo principal mostrar a viabilidade de utilização do ASDA-MRIP, ou seja, validar sua utilização.

Além dessa validação, os resultados obtidos nos testes forneceram dados para uma análise referente à utilização da abordagem MRIP no ambiente proposto. No entanto, os testes foram ampliados de forma a fornecer mais resultados e permitir uma análise mais detalhada.

Conforme descrito na seção 6.2, foram realizados 60 testes para os modelos já descritos nas seções 6.4.1 e 6.4.2 e para mais um modelo. O terceiro modelo é uma cópia do modelo do sistema distribuído, porém com o acréscimo de uma instrução de repetição (instrução *for*) antes do escalonamento do cliente na máquina *lasdix*. O objetivo dessa repetição é aumentar a granulação do processo, permitindo uma análise da influência da granulação na abordagem MRIP. O código do programa de simulação desse terceiro modelo pode ser observado no Apêndice C, figura C.6.

Para os testes realizados na fase de análise foi utilizada uma máquina virtual composta

por 12 máquinas: lasdpc01, lasdpc02, lasdpc03, lasdpc04, lasdpc05, lasdpc06, lasdpc07, lasdpc09, lasdpc10, lasdpc12, lasdpc13 e lasdix.

A análise apresentada nesta seção é abordada tendo como foco a influência do número de replicações no tempo de execução da simulação. A simulação é encerrada quando o nível de confiança e a precisão definidos pelo usuário são atingidos.

Para a análise da influência do número de replicações no tempo de execução de simulação, vários fatores devem ser analisados, tais como:

1. granulação do processo: a granulação do processo está diretamente ligada ao tamanho do modelo a ser simulado. Quanto maior o modelo, mais complexo o programa de simulação e consequentemente maior a granulação do processo;
2. número de processadores: número de computadores que fazem parte da máquina virtual onde a simulação será replicada;
3. arquitetura homogênea ou heterogênea: a arquitetura onde a simulação será executada pode ser homogênea, onde os computadores possuem capacidades computacionais bem semelhantes, ou heterogênea, onde os computadores apresentam capacidades computacionais com grandes variações;
4. método de análise: dependendo do método de análise, alguns modelos fornecem dados que satisfazem as condições de um determinado método mais rapidamente que nos outros métodos;
5. nível de confiança e precisão: informações estipuladas pelo usuário, determinam o grau de precisão e o nível de confiança dos resultados da simulação. Para altos níveis de confiança e precisão o tempo de execução da simulação será mais alto;

Esses fatores não são mutuamente exclusivos, ou seja, cada um deles não possui influência isolada no tempo de execução da simulação. Na maioria das vezes a influência do número de replicações no tempo de simulação depende da combinação de outros fatores. Por exemplo, o número de replicações é influenciado ao mesmo tempo pela granulação do processo (fina, média ou grossa), pelo tipo da arquitetura (se homogênea ou heterogênea) e pelo método de análise.

A seguir será feita uma análise desses fatores tendo como base os testes realizados.

6.5.1 Influência do número de replicações no tempo de execução da simulação

Como destacado no início desta seção, vários são os fatores que devem ser analisados para determinar a influência do número de replicações sobre o tempo de execução de simulação, tais como: granulação do processo, número de processadores, arquitetura homogênea ou heterogênea, método de análise e nível de confiança e precisão. A análise

desenvolvida nesta seção fixa-se separadamente nos fatores (1) granulação do processo e (4) método de análise devido à configuração dos testes realizados. Mesmo considerando nos testes apenas dois fatores, algumas conclusões a respeito dos outros também podem ser obtidas a partir da experiência obtida com os testes de validação.

6.5.1.1 Granulação do processo

De modo a determinar a influência do número de replicações no tempo de execução da simulação, o primeiro fator analisado foi a granulação do processo. A granulação do processo está diretamente relacionada com o tamanho do modelo simulado: modelos mais simples possuem granulação mais fina e modelos mais complexos possuem granulação mais grossa.

O resultado esperado dessa análise é o de que, conforme aumenta-se o número de replicações que precisam ser executadas concorrentemente na mesma máquina e quanto mais complexo é o modelo, maiores serão os tempos de execução da simulação. Isso se deve principalmente à própria definição da abordagem MRIP: o modelo não é particionado e uma (ou mais) cópia(s) do programa de simulação é(são) executada(s) em cada máquina. Se os processos são executados concorrentemente nas máquinas e se o programa de simulação possui uma granulação grossa, cada simulação avança mais lentamente e demora mais para começar a contribuir para com a média final.

De modo a comprovar a influência desse fator e a dimensionar essa influência, foram analisados os tempos de execução das simulações descritas na tabela 6.2 para três modelos: o M/M/1 (granulação mais fina), o do sistema distribuído (granulação média) e o do sistema distribuído com o acréscimo do comando de repetição (granulação mais grossa). Para cada modelo, foram consideradas 12 replicações (1 replicação/processador), 24 replicações (2 replicações/processador) e 36 replicações (3 replicações/processador). Para cada uma dessas situações foram coletados 10 tempos, cada um utilizando uma semente diferente. A importância em dimensionar a influência do número de replicações executando em um mesmo processador está principalmente no fato de que quanto maior o número de replicações executadas, maior a probabilidade de se ter uma replicação contribuindo para a obtenção do resultado final em um tempo menor. Para o caso em que se tem poucos processadores disponíveis, a execução concorrente de diversas replicações, apesar de ser mais lenta, pode ser importante.

Nas tabelas 6.8, 6.9 e 6.10 podem ser observados os tempos de execução da simulação obtidos, respectivamente, nas execuções do modelo M/M/1, sistema distribuído e sistema distribuído com loop, utilizando o método Análise Espectral. Nas tabelas 6.11, 6.12 e 6.13 podem ser observados os tempos de execução da simulação obtidos, respectivamente, nas execuções do modelo M/M/1, sistema distribuído e sistema distribuído com loop, utilizando o método Padronização de Séries Temporais.

Tabela 6.8. Tempo de simulação (segundos) do modelo M/M/1 e método Análise Espectral

	1 replicação	2 replicações	3 replicações
Semente 100	0,282634	0,556818	0,687928
Semente 1578	0,569248	0,726417	0,820321
Semente 2198	0,447173	0,613514	0,814778
Semente 3333	0,402766	0,700626	0,742893
Semente 500	0,487563	0,660677	1,081826
Semente 5279	0,487615	0,620824	0,829862
Semente 537	0,384056	0,660634	0,877442
Semente 6543	0,38602	0,75302	1,090258
Semente 8484	0,377962	0,509303	1,241689
Semente 9765	0,933325	0,48204	1,239759
media	0,4758362	0,6283873	0,9426756
delta	0,11084439	0,055988915	0,125996208

Tabela 6.9. Tempo de simulação (segundos) do modelo sistema distribuído e método Análise Espectral

	1 replicação	2 replicações	3 replicações
Semente 100	3,537021	5,142801	6,823146
Semente 1578	4,504919	7,115334	7,05424
Semente 2198	3,243754	5,094779	7,196008
Semente 3333	3,431208	4,668015	6,255293
Semente 500	2,751719	5,355534	6,191834
Semente 5279	3,502658	4,366107	6,846821
Semente 537	2,580747	5,353021	6,832185
Semente 6543	4,101566	5,236961	6,639106
Semente 8484	5,033703	5,231201	7,548978
Semente 9765	4,388641	5,456169	8,159709
media	3,7075936	5,3019922	6,954732
delta	0,48712659	0,446594881	0,363050971

Tabela 6.10. Tempo de simulação (segundos) do modelo sistema distribuído com repetição e método Análise Espectral

	1 replicação	2 replicações	3 replicações
Semente 100	13,109263	16,465174	20,634592
Semente 1578	11,607743	17,556374	21,961227
Semente 2198	9,973193	16,657246	21,682924
Semente 3333	8,918437	13,640476	23,420496
Semente 500	8,228643	15,917261	20,760778
Semente 5279	9,932667	15,510879	20,607058

	1 replicação	2 replicações	3 replicações
Semente 537	10,390589	15,899485	20,384806
Semente 6543	16,943209	17,350702	20,916708
Semente 8484	14,081243	17,399567	22,800463
Semente 9765	12,030187	15,297318	22,882456
media	11,5215174	16,1694482	21,6051508
delta	1,63400442	0,741569544	0,688406579

Tabela 6.11. Tempo de simulação (segundos) do modelo M/M/1 e método Padronização de Séries Temporais

	1 replicação	2 replicações	3 replicações
Semente 100	0,676634	0,980395	2,049541
Semente 1578	0,799463	1,591318	0,967069
Semente 2198	0,758543	2,190743	1,778778
Semente 3333	1,258613	1,788997	1,802347
Semente 500	1,08291	2,131404	1,248123
Semente 5279	1,082086	1,824802	1,465822
Semente 537	2,341715	2,032393	1,369288
Semente 6543	1,654903	0,852229	2,184604
Semente 8484	1,341758	1,188842	1,962111
Semente 9765	0,509463	1,379171	2,857339
media	1,1506088	1,5960294	1,7685022
delta	0,33588972	0,296911852	0,335713961

Tabela 6.12. Tempo de simulação (segundos) do modelo sistema distribuído e método Padronização de Séries Temporais

	1 replicação	2 replicações	3 replicações
Semente 100	7,149098	19,153538	12,808232
Semente 1578	8,435169	7,520829	10,996219
Semente 2198	4,491034	7,934537	22,402746
Semente 3333	3,849558	6,331432	9,09613
Semente 500	2,801339	5,228911	18,903711
Semente 5279	5,570748	10,457687	17,887968
Semente 537	8,042069	13,710085	27,777882
Semente 6543	5,486532	24,567936	31,956266
Semente 8484	13,181927	14,6723	26,625542
Semente 9765	19,411913	32,220367	31,090307
media	7,8419387	14,1797622	20,9545003
delta	3,10951216	5,456266741	5,162262209

Tabela 6.13. Tempo de simulação do modelo sistema distribuído com repetição e método Padronização de Séries Temporais

	1 replicação	2 replicações	3 replicações
Semente 100	19,605017	39,378902	50,588352
Semente 1578	21,262026	40,238487	47,848534
Semente 2198	15,261477	24,039375	44,94128
Semente 3333	27,674742	42,731998	57,896847
Semente 500	9,771215	18,216282	35,3685
Semente 5279	26,721054	52,524113	62,102497
Semente 537	24,678507	40,473389	58,972141
Semente 6543	19,609852	39,219814	44,6922
Semente 8484	24,605185	34,234432	61,600632
Semente 9765	35,911625	42,599762	63,588013
media	22,51007	37,3656554	52,7598996
delta	4,46140924	6,077871965	5,86238957

As tabelas fornecem além dos tempos de execução, também os valores da média e delta (nível de confiança 95%) dos tempos. A inclusão desses valores tem como objetivo facilitar a análise e a visualização das conclusões. Os gráficos das figuras 6.5, 6.6 e 6.7 apresentam os intervalos de confiança (média – delta; média + delta) dos tempos de simulação (eixo X) em função do número de replicações/processador (eixo Y), para os modelos M/M/1, sistema distribuído e sistema distribuído com repetição, respectivamente, todos utilizando o método Análise Espectral. Os gráficos das figuras 6.8, 6.9 e 6.10 apresentam os intervalos de confiança (média – delta; média + delta) dos tempos de simulação (eixo X) em função do número de replicação/processador (eixo Y), para os modelos M/M/1, sistema distribuído e sistema distribuído com repetição, respectivamente, todos utilizando o método Padronização de Séries Temporais.

Figura 6.5. Intervalos de confiança dos tempos de simulação do modelo M/M/1 utilizando o método Análise Espectral

Figura 6.6. Intervalos de confiança dos tempos de simulação do modelo Sist. Distribuído utilizando o método Análise Espectral

Figura 6.7. Intervalos de confiança dos tempos de simulação do modelo Sist. Distribuído com repetição utilizando o método Análise Espectral

Figura 6.8. Intervalos de confiança dos tempos de simulação do modelo M/M/1 utilizando o método Padronização de Séries Temporais

Figura 6.9. Intervalos de confiança dos tempos de simulação do modelo Sist. Distribuído utilizando o método Padronização de Séries Temporais

Figura 6.10. Intervalos de confiança dos tempos de simulação do modelo Sist. Distribuído com repetição utilizando o método Padronização de Séries Temporais

Para o modelo M/M/1 (figuras 6.5 e 6.8), que é o mais simples, não se pode afirmar estatisticamente que um número menor de replicações por processador resulta em um tempo menor de execução da simulação. O que pode ser observado é que a média dos tempos aumenta conforme aumenta-se o número de replicações/processador, no entanto, os intervalos de confiança estão sobrepostos.

No modelo do sistema distribuído (figuras 6.6 e 6.9), onde a granulação já é um pouco mais grossa, observa-se que para o método Análise Espectral (figura 6.6) já pode-se afirmar que quanto menor o número de replicações por processador, menor é o tempo de execução da simulação. No entanto, para o método Padronização de Séries Temporais (figura 6.9) ainda não se pode estatisticamente afirmar que isso é verdade. Novamente, observa-se que as

médias dos tempos de simulação são maiores quando se tem mais replicações por processador, mas os intervalos de confiança ainda estão sobrepostos.

No último modelo, onde acrescentou-se uma repetição no programa de simulação do modelo do sistema distribuído com o objetivo de aumentar a granulação do processo, já se pode estatisticamente afirmar observando as figuras 6.7 e 6.10 que um número menor de replicações por processador sempre resultará em um tempo menor de execução da simulação.

Com esses dados e gráficos pode-se concluir que a granulação do processo é realmente um dos fatores que fazem o número de replicações por processador influenciar no tempo de execução da simulação. Para modelos simples não existe muita influência pois os processos que compartilham o mesmo processador são executados de maneira rápida. Conforme a complexidade do modelo, e consequentemente a granulação do processo, vai aumentando, a influência vai ficando cada vez mais clara. Os processos que compartilham o mesmo processador começam a demorar mais para serem executados e o resultado final da simulação demora mais para ser obtido.

Desta forma, os resultados obtidos mostram que apesar da média para uma replicação ser sempre menor que para duas ou três replicações, em muitas execuções pode-se obter resultados em um tempo menor ativando duas ou até mesmo três replicações por processador.

Analisando-se ainda os resultados apresentados, pode-se observar que a utilização de duas (ou três) replicações não implica na multiplicação do tempo de execução por dois (ou três). A tabela 6.14 mostra para cada modelo e para cada método a variação do tempo de execução em relação ao tempo de execução para aquele modelo com uma replicação. Observa-se ainda na tabela 6.14 que o método Padronização de Séries Temporais (PST) apresenta uma maior influência no número de replicações por processador que o método Análise Espectral (AE). Essa maior influência é devido à própria definição do método Padronização de Séries Temporais, a qual será explicada na próxima seção.

Tabela 6.14. Variação do tempo de execução em relação ao tempo de execução com uma replicação

Método Nº replicações Modelo	AE			PST		
	1	2	3	1	2	3
M/M/1	1	1,31	1,95	1	1,39	1,53
Sist. Distr	1	1,43	1,85	1	1,81	2,68
Sist. Distr 2	1	1,40	1,87	1	1,66	2,34

6.5.1.2 Método de análise e característica da arquitetura

Além do fator granulação do processo, outros três fatores também estão relacionados com a influência do número de replicações por processador no tempo de execução da simulação. São eles: o método de análise utilizado, o número de máquinas e se essas máquinas compõem uma máquina virtual heterogênea ou homogênea. Os dois últimos fatores estão relacionados à arquitetura do sistema onde a simulação é executada.

Para a análise desses fatores, os dados fornecidos na seção 6.5.1.1, nas tabelas 6.5, 6.6, 6.7, 6.8, 6.9 e 6.10 foram reagrupados em gráficos de modo a facilitar a comparação entre os métodos Análise Espectral e Padronização de Séries Temporais. O gráfico da figura 6.11 representa as médias e os intervalos de confiança das execuções do modelo M/M/1, o da figura 6.12 do modelo sistema distribuído e o da figura 6.13 do modelo sistema distribuído com repetição.

Como pode ser observado, existe um aumento de mais de 100% nos tempos de execução utilizando o método Padronização de Séries Temporais em comparação com os tempos obtidos utilizando o método Análise Espectral. Para todos os casos, quando se compara o mesmo modelo e o mesmo número de replicações, o método Análise Espectral é estatisticamente mais rápido que o Padronização de Séries Temporais.

Figura 6.11. Intervalos de confiança dos tempos de execução da simulação do modelo M/M/1 utilizando os métodos Análise Espectral e Padronização de Séries Temporais

Figura 6.12. Intervalos de confiança dos tempos de execução da simulação do modelo sistema distribuído utilizando os métodos Análise Espectral e Padronização de Séries Temporais

Figura 6.13. Intervalos de confiança dos tempos de execução da simulação do modelo sistema distribuído com repetição utilizando os métodos Análise Espectral e Padronização de Séries Temporais

Além dessa diferença, o que pode ser observado é que os tempos obtidos utilizando Análise Espectral não possuem uma grande variabilidade (intervalo de confiança menor) enquanto que os obtidos com a utilização do método Padronização de Séries Temporais apresenta uma variabilidade bem maior (intervalo de confiança maior).

Uma explicação para essa diferença de variabilidade entre os dois métodos deve-se ao fato da própria definição do método Padronização de Séries Temporais. Neste método, para que uma replicação satisfaça as condições para participar da média final, as variáveis aleatórias definidas na fórmula 2.20 (capítulo 2) devem obedecer a uma distribuição normal. Nem todas as replicações conseguem obedecer a essa condição e, consequentemente estas replicações não participam da média final. Como não são muitas replicações participando, o tempo necessário para se alcançar a precisão estipulada se torna maior.

Desse modo, pode-se observar que o método Padronização de Séries Temporais possui uma dependência maior do número de replicações e das características das máquinas que fazem parte da máquina virtual.

O método Padronização de Séries Temporais é dependente do número de replicações pois quanto maior esse número, maiores são as chances de mais replicações participarem da média final. Quando se tem disponível um número razoável de máquinas para que a simulação seja executada aconselha-se utilizar somente uma replicação por processador. No entanto, caso se tenha um número pequeno de máquinas aconselha-se utilizar mais de uma replicação por processador com o objetivo de aumentar o número de replicações com chances de participar da média final.

O método Padronização de Séries Temporais é dependente das características da máquina virtual pois em redes com máquinas heterogêneas existe a possibilidade das replicações que virão a participar da média estarem sendo executadas em máquinas com baixo poder computacional, aumentando assim o tempo de execução da simulação. Quanto mais homogênea for a máquina virtual, menor será a variabilidade nos tempos de execução da simulação.

O método Análise Espectral não apresenta tanta variabilidade nos tempos de execução pois neste método, as replicações estarão habilitadas a enviar suas contribuições para o mestre de acordo com o tamanho do período transiente. Desse modo, o número de replicações que colaboram com a média final não varia muito e quase todas as replicações conseguem participar. Neste caso, o fato da máquina virtual ser heterogênea ou homogênea apresenta pouca influência.

6.5.2 Conclusões sobre a análise dos resultados obtidos

Na análise realizada com os resultados obtidos a partir dos testes pode-se observar algumas diretrizes para a utilização da abordagem MRIP. Resumidamente, essas diretrizes dizem que:

- Para modelos simples, o número de replicações que deve ser executado em cada máquina não influencia muito o tempo de execução da simulação;

- Para modelos mais complexos, deve-se evitar a execução de mais de uma replicação por processador pois a granulação do processo e a concorrência fazem com que cada replicação seja executada mais lentamente;
- Pode ser necessário que o número de replicações por processador seja maior do que um caso o número de máquinas que fazem parte da máquina virtual seja pequeno e se esteja utilizando o método Padronização de Séries Temporais;
- Para máquinas virtuais muito heterogêneas deve-se utilizar o método Análise Espectral pois este apresenta uma menor variabilidade no tempo de execução

6.6 Considerações Finais

Sendo o ASDA especificado no capítulo 4 e tendo a implementação de 3 de seus módulos descrito no capítulo 5, este capítulo mostrou a viabilidade de uso do ambiente ASDA-MRIP e também provou algumas das conclusões apresentadas no capítulo 4, na especificação do módulo Avaliador.

A viabilidade de utilização do ASDA-MRIP foi mostrada em duas fases: em uma primeira fase os resultados fornecidos foram comparados com resultados obtidos através da resolução de modelos analíticos e em uma segunda fase foi comprovado que os resultados fornecidos satisfazem o nível de confiança e precisão estipulados pelo usuário.

Para a fase de validação foram simulados dois modelos: um sistema M/M/1 e um Sistema Computacional Distribuído. Para cada modelo foram definidas 32 execuções, as quais tinham como variáveis o método de análise, a semente inicial, a quantidade de replicações e o nível de confiança. Para a fase de análise foram simulados três modelos: o M/M/1, o sistema distribuído e o sistema distribuído com uma instrução de repetição, definindo 60 execuções para cada modelo.

Na fase de análise dos resultados diversas características foram consideradas, dentre elas a influência do número de replicações por processador no tempo de execução da simulação, método de análise e o nível de confiança.

Com relação à influência do número de replicações por processador no tempo de execução da simulação pode-se concluir que depende da combinação de vários fatores, tais como: granulação do processo, número de replicações, método de análise e característica da arquitetura onde a simulação é executada.

O próximo capítulo tem como objetivo apresentar as conclusões gerais obtidas com o desenvolvimento desta tese.

Conclusões

7.1 Conclusões Gerais

O trabalho desenvolvido e detalhado nesta tese de doutorado especifica um ambiente automático para desenvolvimento de simulação distribuída, ambiente este denominado ASDA – Ambiente de Simulação Distribuída Automático. Este ambiente oferece ao usuário diversas características que têm como objetivo principal facilitar o desenvolvimento de simulação distribuída. Além da especificação, houve também a implementação de um protótipo do ambiente e, com a análise dos resultados obtidos a partir de um dos módulos do protótipo, pode-se chegar a algumas conclusões sobre a utilização de uma das abordagens para simulação distribuída, a abordagem MRIP.

O conceito de simulação distribuída utilizada nesta tese é o que tem como objetivo a avaliação de desempenho de sistemas computacionais. Foi destacado no capítulo 2 um outro conceito de simulação distribuída que vem sendo estudado: a simulação distribuída interativa, na qual a simulação é executada em diversas máquinas e em tempo real. Esse conceito de simulação é pesquisado e desenvolvido principalmente pela área militar para simulações de combates e guerras.

A necessidade de desenvolvimento de ambientes que tenham como objetivo facilitar a utilização de simulação distribuída deve-se principalmente à dificuldade encontrada pelo usuário em todo o processo de desenvolvimento da simulação. São necessários conhecimentos referentes a modelagem, simulação, análise estatística, computação paralela e finalmente simulação distribuída. Todos os passos e as dificuldades encontradas no desenvolvimento de simulação distribuída foram relatados no decorrer dos capítulos 2 e 3 desta tese.

Uma primeira especificação do ASDA foi feita de maneira informal, através de um diagrama modular composto por sete módulos, sendo que cada um desses módulos possui funções específicas. Uma especificação mais formal do ambiente foi realizada com o auxílio da UML (*Unified Modelling Language*) através do desenvolvimento de três diagramas: o de casos de uso, de classes e de atividades. A especificação do ASDA foi descrita no capítulo 4 desta tese.

O ASDA foi especificado para que possa ser utilizado tanto por usuários sem muita experiência, quanto por usuários com experiência, mas que queiram minimizar o tempo necessário para o desenvolvimento de programas de simulação. Se necessário, o usuário mais experiente tem no ASDA todas as funcionalidades para o refinamento do código gerado. A especificação do modelo pode ser realizada através de duas abordagens: Redes de Filas e

Queueing Statecharts. A escolha destas técnicas foi feita a partir de estudos das diferentes abordagens disponíveis na literatura e de resultados obtidos em diversos trabalhos desenvolvidos no grupo de Sistemas Distribuídos e Programação Concorrente do ICMC – USP.

Além de desenvolver uma simulação distribuída desde o início (especificando o modelo, gerando o código, escolhendo a abordagem a ser utilizada na geração do programa de simulação, etc.), o usuário pode utilizar uma outra característica do ambiente: a possibilidade de diminuir o tempo necessário para se obter resultados confiáveis de um programa de simulação já desenvolvido, replicando-o com a abordagem MRIP. Além de diminuir o tempo de execução da simulação, esta abordagem disponível no ASDA também elimina os problemas de desenvolvimento de um novo programa para um modelo que já possuía um programa desenvolvido. Um programa de simulação sequencial existente pode ser replicado pelo ambiente, que ainda oferece todas as ferramentas necessárias para a obtenção de resultados confiáveis.

Uma característica importante no ASDA é a capacidade que o ambiente tem de fornecer ao usuário um auxílio no desenvolvimento da simulação. Esse auxílio é fornecido tanto quando o usuário especifica o modelo quanto na fase de geração automática do código. Na especificação tem-se a verificação e validação do modelo e dos parâmetros, sendo que o ambiente avisa quando o modelo apresenta inconsistências.

O auxílio na fase de geração do código de simulação é a parte mais complexa do ambiente. O objetivo dessa fase é o de gerar o código do programa de simulação distribuída, porém considerando o fator desempenho da simulação. O programa de simulação pode ser elaborado segundo as abordagens SRIP e MRIP e, caso seja escolhida a SRIP, deve-se ainda optar entre os protocolos conservativos ou otimistas. Vários trabalhos encontrados na literatura tratam da questão de análise de desempenho das abordagens e dos protocolos sendo que vários são os fatores que influenciam esse desempenho. A especificação do módulo que proporciona esse auxílio foi definida tendo como base as conclusões desses trabalhos, dos quais pode-se destacar os desenvolvidos do próprio grupo de Sistemas Distribuídos e Programação Concorrente do ICMC – USP.

Tendo o ASDA sido especificado, houve a necessidade de implementação de um protótipo, o qual é descrito no capítulo 5. Para o desenvolvimento desse protótipo os esforços foram concentrados na implementação de três módulos do ASDA: o módulo de Interface com o Usuário, o módulo Replicador e o módulo de Interface de Software. O módulo de Interface com o Usuário foi escolhido pois sem ele o usuário não tem como interagir com o ambiente e também especificar o modelo. O módulo Replicador foi escolhido pois, dentro da área de pesquisa desenvolvida no grupo, a abordagem MRIP ainda não havia sido pesquisada. Para o módulo de Interface de Software foi definida a interface para 3 bibliotecas: SMPLX, SimPack e SSS.

O desenvolvimento do módulo Replicador, que baseia-se na implementação do ASDA-MRIP, possibilitou a execução de alguns modelos com o objetivo principal de validar o ambiente desenvolvido. A validação ocorreu em fases: uma primeira comprovou que os resultados obtidos estavam de acordo com o nível de confiança e precisão estipulados pelo usuário e uma segunda comprovou que os resultados obtidos eram os esperados. Além de validar o ASDA-MRIP, os testes foram ampliados de modo que fosse possível obter algumas conclusões sobre a utilização da abordagem MRIP. O enfoque dado nos testes foi a influência no tempo de simulação do número de replicações que estão sendo executadas concorrentemente em cada máquina.

7.2 Contribuições desta Tese

O trabalho descrito nesta tese aborda vários aspectos relacionados ao desenvolvimento de simulação distribuída. Dentre esses aspectos, destaca-se modelagem, abordagens para simulação distribuída (SRIP e MRIP), partição do modelo em processos lógicos, desenvolvimento do programa de simulação e análise estatística dos resultados obtidos na simulação.

O desenvolvimento da simulação se torna mais fácil a partir do momento que o usuário tem ao seu dispor um ambiente onde possa especificar seu modelo, gerar o código do programa de simulação utilizando uma abordagem que lhe proporcionará o melhor desempenho, obter resultados confiáveis, sendo tudo realizado de uma maneira gráfica e amigável.

As principais contribuições deste doutorado são:

- Realização de uma análise crítica da bibliografia existente, fornecendo a contextualização do conceito de simulação distribuída utilizado neste trabalho (ferramenta para avaliação de desempenho);
- Levantamento na literatura dos ambientes de simulação que têm como objetivo facilitar o desenvolvimento de simulação, tanto sequencial como distribuída;
- Identificação das principais dificuldades encontradas no desenvolvimento de simulações sequenciais e distribuídas;
- Especificação de um ambiente para desenvolvimento de simulação distribuída (ASDA), o qual apresenta uma série de características que não são encontradas em ambientes que possuem o mesmo objetivo do ambiente proposto. Dentre essas características, pode-se citar:
 - Possibilidade de especificação do modelo utilizando duas abordagens: Redes de Filas e *Queueing Statecharts*. Para a definição das abordagens a serem utilizadas no ASDA foi realizada uma compilação de diversos

trabalhos que comparam as diversas abordagens existentes. Essa compilação, apresentada no capítulo 2, e a análise efetuada, que levou a escolha de Redes de Filas Estendidas (por ser uma abordagem amplamente conhecida pelos usuários de simulação) e *Queueing Statecharts* (por oferecer maior flexibilidade) é mais uma contribuição deste trabalho;

- Facilidade para inserir novas abordagens para especificação de modelos. Como a especificação do ambiente foi realizada utilizando o conceito de orientação a objetos, esta apresenta uma maneira mais fácil de acrescentar novas classes que representem outras formas de modelagem;
- Geração automática do código do programa de simulação distribuída. Essa característica permite que o usuário não precise se preocupar com os conceitos de programação e principalmente, em se tratando de simulação distribuída, dos conceitos de comunicação e sincronização;
- Auxílio do ambiente na fase de especificação do modelo, não permitindo que o usuário especifique de forma inconsistente. A inconsistência pode estar na definição do modelo ou na parametrização;
- Possibilidade de utilização das abordagens seqüencial, SRIP, MRIP ou SRIP + MRIP. A escolha de qual abordagem utilizar pode ficar a cargo do usuário ou então pode ser feita com o auxílio do ambiente. O ambiente analisa o modelo e sua parametrização, a plataforma onde a simulação será executada e decide pela abordagem que fornece o melhor desempenho;
- Possibilidade de utilização de um programa de simulação seqüencial já existente. Para essa característica, o ambiente fornece interfaces para diversas linguagens/bibliotecas de simulação.

A especificação do ASDA proposta nesta tese foi formalizada utilizando a UML, através dos diagramas de casos de uso, de classes e de atividades. Esta forma de especificação facilita a implementação e a manutenção do ambiente proposto;

- Desenvolvimento de um protótipo para o ambiente proposto (ASDA). Para validar as idéias presentes na especificação do ASDA, foi desenvolvido um protótipo, incluindo três módulos: Interface com o Usuário, Replicador e Interface de Software.
- Desenvolvimento de uma biblioteca que permite o desenvolvimento de simulações utilizando a abordagem MRIP (ASDA-MRIP). Nesta biblioteca estão disponíveis três métodos para análise de saída e o usuário pode definir qual o nível de

confiança e precisão desejado.

- Resultados obtidos com a execução de modelos computacionais proporcionaram a obtenção de algumas conclusões sobre a utilização da abordagem MRIP sendo analisado a influência do número de replicações no tempo de execução da simulação segundo os fatores granulação do processo, método de análise e arquitetura.
- Definição de regras para a utilização das abordagens MRIP e SRIP e também dos protocolos conservativos e otimistas;
- As publicações geradas tendo como base os resultados obtidos no desenvolvido desta tese são:
 - Aceitas:
 - BRUSCHI, S. M.; SANTANA, R. H. C.; SANTANA, M. J. ASDA – An Automatic Distributed Simulation Environment. In: *2000 Summer Computer Simulation Conference, (SCSC' 2000)* Vancouver – Canada. July 16-20 2000, Proceedings.
 - BRUSCHI, S. M.; ULSON, R.S.; SANTANA, R. H. C.; SANTANA, M. J. Simulation as a Tool for Computer Architecture Teaching. In: *1999 Summer Computer Simulation Conference, (SCSC' 1999)* Chicago – Illinois – July 11-15 1999, Proceedings.
 - A serem submetidas
 - BRUSCHI, S. M.; SANTANA, R. H. C.; SANTANA, M. J. The specification of an Automatic Distributed Simulation Environment - a ser submetido para uma revista indexada internacional.
 - BRUSCHI, S. M.; SANTANA, R. H. C.; SANTANA, M. J. Analysing the utilization of MRIP simulation - a ser submetido ao ACM SIGMETRICS 2003

7.3 Sugestões para Trabalhos Futuros

A especificação do ambiente automático para desenvolvimento de simulação distribuída proposto nesta tese constitui uma definição inicial de um ambiente que pode expandir muito mais e agregar novas características. Algumas dessas expansões, apresentadas como sugestões para trabalhos futuros, são:

- Detalhamento maior da especificação formal do ASDA através da definição de

outros diagramas UML, tais como os diagramas de colaboração, componentes, interação e implantação;

- Definição de regras para interoperabilidade entre modelos desenvolvidos com *Queueing Statecharts* e Redes de Filas;
- Realização de um conjunto de testes comparativos utilizando o ASDA-MRIP, o SMPLX (biblioteca que utiliza o protocolo conservativo CMB) e o Warped (linguagem que utiliza o protocolo otimista Time Warp);
- Ampliação dos testes realizados com o objetivo de analisar a utilização da abordagem MRIP, levando em consideração outros fatores, tais como: a característica da arquitetura (homogênea ou heterogênea), o método de análise, o número de processadores utilizado e o número de replicações por processador;
- Ampliação das regras do módulo Avaliador com base nos testes citados acima e implementação dessas regras utilizando lógica fuzzy;
- Inclusão de mais bibliotecas/linguagens nas definições do módulo Gerador;
- Inclusão de mais bibliotecas/linguagens nas definições do módulo Interface de Software;
- Expansão do ASDA-MRIP para análise de mais de uma variável ao mesmo tempo;
- Inclusão de recursos para controle de replicações em simulações SRIP;
- Definição de recursos para inclusão de desenvolvimento de projeto experimental e análise dos resultados de um conjunto de experimentos.

Referências Bibliográficas

- (Alexopoulos & Seila, 1998) ALEXOPOULOS, C.; SEILA, A. F. Output Data Analysis. In: BANKS, J. *Handbook of Simulation*. Georgia, Atlanta: John Wiley & Sons, Inc., p. 225-272, 1998.
- (Alexopoulos, 1994) ALEXOPOULOS, C. A review of advanced methods for simulation output analysis. In: *Proceedings of the 1994 Winter Simulation Conference (WSC'94)*, New Jersey - USA, IEEE, p. 133-140, 1994.
- (Almasi & Gottlieb, 1994) ALMASI, G. S., GOTTLIEB, A., *Highly Parallel Computing*, 2^a ed., The Benjamin Cummings, 1994.
- (Araujo, 1999) ARAUJO, A. P. F. DPWP: Uma Nova Abordagem ao Escalonamento Dinâmico em Computação Paralela Virtual. São Carlos, 1999. *Dissertação (Mestrado em Ciências da Computação e Matemática Computacional)* - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.
- (Ayani, 1993) AYANI, R. Parallel Simulation. In: *Performance Evaluation of computer and communication systems, Lecture Notes in Computer Science 729*, Springer-Verlag, p. 1-20, 1993.
- (Bagrodia & Takai, 2000) BAGRODIA, R.; TAKAI, M. Performance Evaluation of Conservative Algorithms in Parallel Simulation Languages. *IEEE Transactions on Parallel and Distributed Systems*, v.11, n.4, p.395-411, 2000.
- (Bagrodia et al., 1998) BAGRODIA, R.; MEYER, R.; TAKAI, M.; CHEN, Y.; ZENG, X.; MARTIN, J.; SONG, H. PARSEC: A Parallel Simulation Environment for Complex Systems. *IEEE Computer*, v.3, n.10, p.77-85, 1998.
- (Balakrishnan et al., 2001) BALAKRISHNAN, V.; RADHAKRISHNAN, R.; RAO, D. M.; ABU-GHAZALEH, N.; WILSEY, P. A. A Performance and Scalability Analysis Framework for Parallel Discrete Event Simulators, *Simulation Practice and Theory*, v. 8, p. 529-553, 2001.
- (Banks, 1998) BANKS, J. Principles of Simulation. In: BANKS, J. *Handbook of Simulation*. Georgia, Atlanta: John Wiley & Sons, Inc., p. 3-30, 1998.
- (Bell & O'Keefe, 1987) BELL, P.C.; O'KEEFE, R. M. Visual Interactive Simulation - History, recent developments, and major issues. *Simulation*, v. 49, n.3, p. 109-116, 1987.
- (Berry & Jefferson, 1985) BERRY, O.; JEFFERSON, D. Critical Path Analysis of Distributed Simulation. In: *Proceeding of SCS Multiconference Distributed Simulation*, California - USA, p. 57-60, 1985.
- (Bizarro et al., 1998) BIZARRO, P.; SILVA, L.M.; SILVA, J.G. JWarp: A Java Library For Parallel Discrete-Event. Simulations *Simulation Practice and Theory*, v. 8, p. 529-553, 2001. *Concurrency: Practice and Experience*, v. 10, p. 999-1005, 1998.

- (Booch et al., 2000) BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. UML: Guia do Usuário Editora Campus, 1ª edição, 2000.
- (Borland, 2001a) BORLAND. RADical Performance Linux Application Development. Borland Evaluators Guide. January 2001.
- (Borland, 2001b) BORLAND. Kylix – Developer's Guide. Borland. 2001.
- (Brennan et al., 1995) BRENNAN, J.E.; WALENTY, M.E.; MOORE, K.E. Simulating a UNIX Processor Using Petri Nets and ALPHA/SIM. In: *Proceeding of the 1995 Summer Computer Simulation Conference (SCSC'95)*, Ottawa - Canada, p.63-69, 1995.
- (Bruschi, 1997) BRUSCHI, S. M. Extensão do ASiA para Simulação de Arquiteturas de Computadores. São Carlos, 1997. *Dissertação (Mestrado em Ciências da Computação e Matemática Computacional)* - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.
- (Bruschi et al., 1999) BRUSCHI, S. M.; ULSON, R. S.; M.; SANTANA, R. H. C.; SANTANA, M. J. Simulation as a Tool for Computer Architecture Teaching. In: *Proceeding of the 1999 Summer Computer Simulation Conference (SCSC' 1999)*, Chicago - USA, July 11-15 1999.
- (Bruschi et al., 2000) BRUSCHI, S.; M.; SANTANA, R. H. C.; SANTANA, M. J. ASDA – An Automatic Distributed Simulation Environment. In: *Proceeding of the 2000 Summer Computer Simulation Conference (SCSC' 2000)*, Vancouver - Canada, July 16-20 2000.
- (Bryant, 1977) BRYANT, R.E. Simulation of Packet Communications Architecture Computer. Massachusetts : Massachusetts Institute of Technology, 1977 (*Relatório Técnico 188 – MIT-LCS*).
- (Cavitt et al., 1996) CAVITT, D. B.; OVERSTREET, C. M.; MALY, K. J. A Performance Analysis Model for Distributed Simulations. In: *Proceeding of Winter Simulation Conference (WSC'96)*, Coronado - USA, December 8-11 1996.
- (Chandy & Misra, 1979) CHANDY, K. M.; MISRA J. 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, 1979.
- (Chandy & Misra, 1981) CHANDY, K. M.; MISRA, J. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of ACM*, v.24, n.11, p.198-205, 1981.
- (Cowie, 1998) COWIE, J.R. JTED: parallel discrete-event simulation in Java. *Simulations Simulation Practice and Theory*, v. 8, p. 529-553, 2001. *Concurrency: Practice and Experience*, v. 10, p. 993-997, 1998.
- (Das, 1996) DAS, S. R. Adaptive Protocols for Parallel Discret Event Simulation . In: *Proceedings of the 1996 Winter Simulation Conference (WSC'96)*, Coronado - USA. p. 186-193, December 8-11 1996.
- (Dota, 2001) DOTA, M. A. Avaliação de Desempenho de uma Ferramenta para Simulação Distribuída Baseada no Protocolo Otimista Time Warp. São Carlos, 2001. *Dissertação*

- (*Mestrado em Ciências da Computação e Matemática Computacional*) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.
- (Ewing *et al.*, 1997) EWING, G C.; MCNICKLE, D.; PAWLIKOWSKI, K. Multiple Replications in Parallel: Distributed Generation of Data for Speeding up Quantitative Stochastic Simulation In: *Proceedings of the 15th Congress of Int. Association for Mathematics and Computer in Simulation (IMAC'97)*, Berlin - Germany, Wissenschaft und Technik Verlag, p. 397-402, Aug 1997.
- (Ewing *et al.*, 1998) EWING, G.C.; PAWLIKOWSKI, K.; MCNICKLE, D. *Akaroa 2.5: User's Manual*, Department of Computer Science, University of Caterbury, New Zeland, September, 1998.
- (Ewing *et al.*, 1999) EWING, G.C.; PAWLIKOWSKI, K.; MCNICKLE, D. Akaroa-2: Exploiting Network Computing by Distributing Stochastic Simulation. In: *Proceeding of European Simulation Multiconference (ESM'99)*, Warson - Poland, p.175-181, 1999.
- (Ferscha, 1995) FERSCHA, A.; TRIPATHI, S.K. Parallel and Distributed Simulation of Discrete Event Simulation. *Parallel and Distributed Computing Handbook*. MacGraw-Hill. Cap.35. 1995.
- (Ferscha & Johnson.,1996) FERSCHA, A; JOHNSON, J.; A Testbed for Parallel Simulation Performance Prediction. In: *Proceedings of Winter Simulation Conference (WCS'96)*, Coronado - USA, December 8-11 1996.
- (Ferscha *et al.*, 2001) FERSCHA, A; JOHNSON, J.; TURNER, S. J. Distributed Simulation Performance Data Mining. *Future Generation Computer Systems*. v.18, p. 157-174, 2001.
- (Figueiredo, 2000) FIGUEIREDO, T. C. Interface AMIGO-MPI: Uma Abordagem Flexível e Dinâmica para Escalonamento de Processos. São Carlos, 2000. *Dissertação (Mestrado em Ciências da Comutação e Matemática Computacional)*, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.
- (Fishman, 1967) FISHMAN, G. S. Problems in the statistical analysis of simulation experiments: the comparision of means and the length of sample records, *Communications of the ACM*, v.10, n.2, p.94-99, February 1967.
- (Fishwick, 1992) FISHWICK, P. SimPack: Getting Started with Simulation Programming in C and C++. In: *Proceeding of Winter Simulation Conference (WSC'92)*, Arlington – USA, December 13-16 1992.
- (Francês, 1998) FRANCÊS, C. R. L. Stochastic feature charts: uma extensão estocástica para os statecharts. São Carlos, 1998. *Dissertação (Mestrado em Ciências da Comutação e Matemática Computacional)*, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.
- (Francês, 2001) FRANCÊS, C. R. L. Statecharts estocásticos e queueing statecharts: novas abordagens para avaliação de desempenho baseadas em especificação statecharts, 2001. *Tese (Doutorado em Ciências da Computação e Matemática Computacional)*, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.

- (Fujimoto, 1990) FUJIMOTO, R.M. Parallel Discrete Event Simulation. *Communications of the ACM*, v.33, n.10, p. 30-53, 1990.
- (Fujimoto, 2000) FUJIMOTO, R. M. *Parallel and Distributed Simulation Systems*, John Wiley & Sons, 2000.
- (Glynn & Heidelberger, 1992) GLYNN, P. W.; HEIDELBERGER, P. Analysis of initial Transient Deletion for parallel Steady-State Simulations. *SIAM J. Sci. Stat. Comput.*, v.13, n.4, p.904-922, 1992.
- (Harel *et al.*, 1987) HAREL, D.; PNUELI, A.; SCHMIDT, J.P.; SHERMAN, R. On the formal semantics of Statecharts. In: *Proceeding of the 2nd IEEE Symposium on Logic in Computer Science*, New York – USA, p.54-64, 1987.
- (Harel *et al.*, 1990) HAREL, D.; LACHOVER, H.; NAAMAD, A.; PNUELI, A.; POLITI, M.; SHERMAN, R.; SHTULL-TRAURING, A.; TRAKHTENBROT, M. STATEMATE: A working Environment for the development of complex reactive systems. *Transactions on Software Engineering*, v.16, n.4, p.403-414, 1990.
- (Harel, 1987) HAREL, D. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, v.8, p.231-274, 1987.
- (Heidelberger, 1986) HEIDELBERGER, P. Statistical Analysis of Parallel Simulations. In: *Proceeding of 1986 Winter Simulation Conference (WSC'96)*, California - USA, p.290-295, 1986.
- (Heidelberger, 1988) HEIDELBERGER, P. Discrete Event Simulation and Parallel Processing: Statistical Properties. *SIAM J. Stat. Comput.*, v.9, n.6, p.1114-1132, 1988.
- (Heidelberger & Welch, 1981) HEIDELBERGER, P.; WELCH, P. D. Adaptive Spectral Methods for Simulation Output Analysis. *IBM J. Res. Develop.* v.25, p. 860-876, 1981.
- (Heuser, 1990) HEUSER, C. A. Modelagem Conceitual de Sistemas: Redes de Petri. In: *V EBAI, Escola Brasileiro-Argentina de Informática*, 5., Nova Friburgo. Nova Friburgo: EBAI, 150p, 1990.
- (Hix & Hartson, 1993) HIX, D.; HARTSON, H. R. *Developing User Interfaces: ensuring usability through product & process* Wiley Professional Computing, 1ª edição, 1993.
- (Jain, 1991) JAIN, R. The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, *Wiley- Interscience*, New York, NY, April 1991.
- (Janneck, 1998) JANNECK, J. W. Generalizing Lookahead – Behavioral Prediction in Distributed Simulation. In: *Workshop on Parallel and Distributed Simulation (PADS'98)*, Alberta - Canada, 1998.
- (Jefferson, 1985) JEFFERSON, D. R. Virtual Time. *ACM Transactions on Programming Languages and Systems*, v.7, n.3, p.404-425, 1985.

- (Jha & Bagrodia, 1996) JHA, V.; BAGRODIA, R. A Performance Evaluation Methodology for Parallel Simulation Protocols In: *Proceeding of the 10th Workshop on Parallel and Distributed Simulation* (PADS'96), Pennsylvania - USA, p. 180-185, May 1996.
- (Kawabata, 1999) KAWABATA, C. L. O. Extensão do Ambiente de Simulação Automático (ASiA) para Simulação de Redes de Computadores. São Carlos, 1999. *Dissertação (Mestrado em Ciências da Computação e Matemática Computacional)* - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.
- (Kawabata, 2002) KAWABATA, C. L. O. Projeto e Implementação do Mecanismo de Troca de Protocolos de Sincronização em Tempo de Execução. São Carlos, 2002. *Monografia de Qualificação (Doutorado em Ciências da Computação e Matemática Computacional)* - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.
- (Kelton, 1997) KELTON, W. D. Statistical Analysis of Simulation Output. In: *Proceeding of the 1997 Winter Simulation Conference* (WSC'97), Georgia - USA, p. 23-30, 1997.
- (Lim et al., 2000) LIM, C.; LOW, Y.; GAN, B.; CAI, W. Implementation Lessons of Performance Prediction Tool for Parallel Conservative Simulation. Springer-Verlag, *Lecture Notes in Computer Science 1900*. In: *Proceeding of the 6th International Euro-Par Conference*, Munich - Germany, p. 189-193, August/September 2000.
- (Lin, 1992) LIN, Y. B. Parallelism Analyzers for Parallel Discrete Event Simulation. In: *ACM Trans. Modeling and Computer Simulation*, v. 2, n. 3, p. 239-264, July 1992.
- (Low et al., 1999) LOW Y.; LIM, C.; CAI, W.; HUANG S.; HSU W.J.; JAIN, S.; TURNER S. Survey of Languages and Runtime Libraries for Parallel Discrete-Event Simulation. *Simulation*, v.72, n.3, p.170-186, 1999.
- (L'Ecuyer, 1998) L'ECUYER, P. Random Number Generation In: BANKS, J. *Handbook of Simulation*. Georgia, Atlanta: John Wiley & Sons, Inc., p. 93-137, 1998.
- (MacDougall, 1987) MACDOUGALL, M.H. *Simulating Computer Systems Techniques and Tools*. The MIT Press, 1987.
- (Maciel et al., 1996) MACIEL, P.R.M.; LINS, R.D.; CUNHA, P.R.F. Introdução às Redes de Petri e Aplicações. *10^a Escola de Computação*, Campinas, 8 a 13 de Julho de 1996.
- (Mascarenhas et al., 1995) MASCARENHAS, E.; REGO, V.; SANG, J. DISplay: A System for Visual-Interaction in Distributed Simulations. In: *Proceeding of the 1995 Winter Simulation Conference* (WSC'95), Virginia - USA, p.698-705, 1995.
- (Mayhew, 1992) MAYHEW, D. J. Principles and Guidelines in Software User Interface Design Prentice Hall PTR, 1^a edição, 1992.
- (Meng, 1999) MENG, X. Distributed Simulations – Issues and Implementations in Clusters of Workstations Environment. *Computer Systems Science and Engineering*, v.14, n.1, p.27-57, 1999.
- (Meyer, 2000) MEYER, P. L. *Probabilidade: Aplicações à Estatística* LTC, 2^a Edição, 2000.

- (Miguel *et al.*, 1998) MIGUEL, J.; ARRUABARRENA, A.; BEIVIDE, R.; FORTES, J.A.B. An evaluation of implementations of the CMB parallel simulation algorithm on distributed memory multicomputers. *Journal of Systems Architecture* n.44, p. 519-545, 1998.
- (Misra, 1986) MISRA, J. Distributed Discrete-Event Simulation. *Computer Surveys*, v.18, n.1, p.39-65, 1986.
- (Moore & Brennan, 1995) MOORE, K. E.; BRENNAN, J. E. Petri Nets and Simulation: a Tutorial. In: *Proceeding of the 1995 Summer Computer Simulation Conference* (SCSC'95), Ottawa - Canada, p.45-50, 1995.
- (Moore & Brennan, 1995a) MOORE, K. E.; BRENNAN, J. E. ALPHA/SIM Simulation Software Tutorial. In: *Proceeding of the 1995 Winter Simulation Conference* (WSC'95), Virginia - USA, p.387-394, 1995.
- (Moreira, 2002) MOREIRA, E. M. Avaliação das Formas de Sincronização nos Programas de Simulação de Sistemas Computacionais Distribuídos. São Carlos, 2002. *Monografia de Qualificação (Doutorado em Ciências da Computação e Matemática Computacional)* - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.
- (Morselli, 2000) MORSELLI, J. C. M. Junior, Um mecanismo para troca de protocolos de sincronização de simulação distribuída em tempos de execução. São Carlos, 2000. *Tese (Doutorado em Ciências “Física Aplicada – opção: Física Computacional”)* - Instituto de Física de São Carlos, Universidade de São Paulo.
- (Mota *et al.*, 1999) MOTA, E.; WOLISZ, A.; PAWLIKOWSKI, K. Sequential Batch Means Techniques for mean value analysis in Distributed Simulation. In: *Proceeding of European Simulation Multiconference* (ESM-99), Warson - Poland, p.129-134, 1999.
- (Mota *et al.*, 2000) MOTA, E.; WOLISZ, A.; PAWLIKOWSKI, K. A Perspective of Batching Methods in a Simulation Environment of Multiple Replications in Parallel. In: *Proceeding of Winter Simulation Conference* (WSC'00), Orlando – USA, December 10-13 2000.
- (Nicol, 1988) NICOL, D. M. Parallel Discrete Event Simulation of FCFS Stochastic Queuing Networks In: *Parallel Programming: Experience with Application, Language and Systems*, ACM SIGPLAN, p. 124-137, July 1988.
- (Oliveira, 1998) OLIVEIRA, L.L. Um Módulo de Análise de Resultados para o ASiA. São Carlos, 1998. *Dissertação (Mestrado em Ciências da Computação e Matemática Computacional)* - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.
- (Overeinder & Sloot, 1995) OVEREINDER, B; SLOOT, P. M. A. Parallel Performance Evaluation through Critical Path Analysis. In: *Lecture Notes in Computer Science*, n. 919, p. 634-639, Spring-Verlag, Berlin, Germany, May 1995.
- (Overeinder *et al.*, 1991) OVEREINDER, B.; HERTZBERGER B.; SLOOT P. Parallel Discrete Event Simulation. In W.J. Withagen, editor, *The Third Workshop Computer*

- Systems, Faculty of Electrical Engineering, Eindhoven University, Eindhoven, The Netherlands, p. 19-30, May 1991.
- (Page, 1999) PAGE, E. H. Beyond Speedup: PADS, the HLA and Web-Based Simulation, PADS'99. *13th Workshop on Parallel and Distributed Simulation*, Georgia - USA, 2-4 May 1999.
- (Pawlikowski, 1990) PAWLIKOWSKI, K. Steady-State Simulation of Queueing Processes: A Survey of Problems and Solutions. *ACM Computing Surveys*, v.22, n.2, p.123-170, 1990.
- (Pedroso, 1996) PEDROSO, W. Jr. Módulo Visual: Uma Interface Gráfica para o Ambiente de Simulação Automático (ASiA). São Carlos, 1996. *Dissertação (Mestrado em Ciências da Computação e Matemática Computacional)* - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.
- (Pegden *et al.*, 1995) PEGDEN, C.D.; SHANNON, R.E.; SADOWSKI, R.P. Introduction to Simulation using SIMAN. 2nd ed., *MacGraw-Hill International Editions*, New York, 1995.
- (Peterson & Chamberlain, 1993) PETERSON, G. D.; CHAMBERLAIN R. D. Exploiting Lookahead in Synchronous Parallel Simulation. In: *Proceedings of the 1993 Winter Simulation Conference (WSC'93)*, California - USA, p. 706-712, 1993.
- (Pollatschek, 1997) POLLATSCHEK, M. A. *Programming Discrete Simulations: Tools for Modelling the Real World*. CMP Books, 1997.
- (Quaglia *et al.*, 1999) QUAGLIA, F.; CORTELESSA, V.; CICIANI, B. Tradeoff Between Sequential and Time Warp Based Parallel Simulation. *IEEE Transactions on Parallel and Distributed Systems*, v.10, n.8, 1999.
- (Reed *et al.*, 1988) REED, D.A.; MALONY, A.D.; McCDREDIE, B.D. Parallel Discrete Event Simulation Using Shared Memory. *IEEE Transactions on Software Engineering*, v.14, n.4, 1988.
- (Reynolds *et al.*, 1989) REYNOLDS, P. F.; WEIGHT C. F.; FIDLER, J. R. Comparative Analyses of Parallel Simulation Protocols. In: *Proceeding of Winter Simulation Conference (WSC'89)*, Washington – USA. December 4-6 1989.
- (Roetzheim, 2001) ROETZHEIM, William. Kylix vs. gcc Development A productivity and maintainability comparison. Borland. 2001.
- (Rönngren *et al.*, 1996) RÖNNGREN, R., LILJENSTAM, M., AYANI, R., MONTAGNAT, J. A Comparative Study of State Saving Mechanisms for Time Warp Synchronized Parallel Discrete Event Simulation, In: *Proceedings of the 29th Annual Simulation Symposium (SS'96)*, Louisiana - USA, April 8-11, 1996.
- (Sanchez, 1999) SANCHEZ, S. M. ABC's of Output Analysis. In: *Proceeding of the 1999 Winter Simulation Conference (WSC'99)*, Georgia - USA, p. 24-32, 1999.

- (Santana, 1990) SANTANA, M.J. An Advanced Filestore Architecture for a Multiple Lan Distributed Computing System. Southampton, 1990. *Tese (Doutorado)*, University of Southampton.
- (Santana, 1990a) SANTANA, R.H.C. Performance Evaluation of Lan-Based File Servers. Southampton, 1990. *Tese (Doutorado)*, University of Southampton.
- (Santana, 1996) SANTANA, R. H. C., SPOLON, Renata; SANTANA, M. J.; SPOLON, Roberta. Graphical User-interface for an Automatic Simulation System. In: *Proceedings of the 1996 Summer Computer Simulation Conference (SCSC'96)*. Oregon - USA, p.139-143, July 21-25 1996.
- (Santana, 1996a) SANTANA, M. J., SPOLON, Roberta; SANTANA, R.H.C.; SPOLON, Renata. Automatic Generation of Discrete-System Simulation Programs. In: *Proceedings of the 1996 Summer Computer Simulation Conference (SCSC'96)*. Oregon - USA, p.133-138, July 21-25 1996.
- (Santos, 2001) SANTOS, R. R. Escalonamento de Aplicações Paralelas: Interface AMIGO-CORBA. São Carlos, 2001. *Dissertação (Mestrado em Ciências da Computação e Matemática Computacional)* - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.
- (Sauer *et al.*, 1980) SAUER, C. H., MACNAIR, E. A., SALZA, S. A Language for extended queueing networks. *IBM Journal of Research and Development*, n. 24, November 1980.
- (Schruben, 1993) SCHRUBEN, L. W. Confidence interval estimation using stardardized time series, *Operations Research*, v. 30, p. 569-590, 1993.
- (Silva, 2000) SILVA, A. F. Modelos de redes de filas para sistemas computacionais distribuídos. Simulação x Métodos Analíticos. São Carlos, 2000. *Dissertação (Mestrado em Ciências da Computação e Matemática Computacional)* - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.
- (Soares, 1992) SOARES, L.F.G. *Modelagem e Simulação Discreta de Sistemas*, Editora Campus Ltda, 1992.
- (Song, 2001) SONG, H. Y. A Probabilistic Performance Model for Conservative Simulation Protocol. In: *Proceeding of the 15th Workshop on Parallel and Distributed Simulation (PADS'01)*, California - USA, 2001.
- (Souza et al., 1999) SOUZA, P. S. L.; Santana, M.J.; Santana R.H.C.; Araújo, A.P.F. PVM and a Viable and Flexible Scheduling. In: *Proceeding of the 11th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'99)*, Massachusetts - USA, November 3-6 1999.
- (Souza, 2000) SOUZA, P. S. L. AMIGO: Uma Contribuição para a Convergência na Área de Escalonamento de Processos. São Carlos, 2000. *Tese (Doutorado em Ciências “Física Aplicada – opção: Física Computacional”)* - Instituto de Física de São Carlos, Universidade de São Paulo.

- (Spolon, 1994) SPOLON, Renata. Um editor gráfico para um ambiente de simulação automático. São Carlos, 1994. *Dissertação (Mestrado em Ciências da Computação e Matemática Computacional)* - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.
- (Spolon, 1994a) SPOLON, Roberta. Um gerador de aplicações para um Ambiente de Simulação Automático. São Carlos, 1994. *Dissertação (Mestrado em Ciências da Computação e Matemática Computacional)* - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.
- (Spolon, 2001) SPOLON, R. Um Método para Avaliação de Desempenho de Protocolos de Sincronização Otimistas para Simulação Distribuída. São Carlos, 2001. *Tese (Doutorado em Ciências “Física Aplicada – opção: Física Computacional”)* - Instituto de Física de São Carlos, Universidade de São Paulo.
- (Steinman, 1992) STEINMAN, J. S. SPEEDES: A Unified Approach to Parallel Simulation. In: *Proceedings of the 6th Workshop on Parallel and Distributed Simulation (PADS’92)*, California - USA . p. 75-84, 1992.
- (Steinman, 1993) STEINMAN, J. S. Breathing Time Warp. In: *7th Workshop on Parallel and Distributed Simulation (PADS’93)*, California – USA. p. 109-118, 1993.
- (Swain, 1995) SWAIN, J.J. Simulation on Survey: Tools for process understanding and improvement. *OR/MS Today*, August 1995.
- (Swet & Drake, 2001) SWET, R. J.; DRAKE, G. R. The Arena Product Family: Enterprise Modeling Solutions. In: *Proceeding of Winter Simulation Conference (WSC 2001)*, Arligton - USA. December 9-12 2001.
- (Swope & Fujimoto, 1987) SWOPE, S. W.; FUJIMOTO, R. M. Optimal Performance of Distributed Simulation Programs. In: *Proceeding of Winter Simulation Conference*, p. 612-617, December 1987.
- (Tanir & Sevinc, 1994) TANIR, O.; SEVINC, S. Defining Requirements for a Standard Simulation Environment. *IEEE Computer*, p.28-34, February 1994.
- (Tatsumi, 2002) TATSUMI, E. S. Análise e Avaliação do uso da ferramenta ParSMPL em um Ambiente de Simulação Distribuída Automático – ASDA. São Carlos, 2002. *Monografia de Qualificação (Mestrado em Ciências da Computação e Matemática Computacional)* - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.
- (Tay *et al.*, 1997) TAY S.; TEO, Y.M.; KONG, S.T. Speculative Parallel Simulation with an Adaptive Throttle Scheme. In: *Proceeding of the 11th Workshop on Parallel and Distributed Simulation (PADS’97)*, Lockenhaus - Austria, p.116-123, 1997.
- (Thondugulam *et al.*, 1999) THONDUGULAM, N. V.; RAO, D. M.; RADHAKRISHNAN, R.; WILSEY, P. A. Relaxing Casual Constraints in PDES In: *13th International Parallel Processing Symposium (IPPS’99)*, 1999.

- (Ulson, 1999) ULSON, R.S. Simulação Distribuída em Plataformas de Portabilidade: Viabilidade de uso e comportamento do Protocolo CMB. São Carlos, 1999. *Tese (Doutorado em Ciências “Física Aplicada – opção: Física Computacional”)* - Instituto de Física de São Carlos, Universidade de São Paulo.
- (Varga, 2001) VARGA, A. The OMNeT++ Discrete Event Simulation System, In: *Proceedings of the European Simulation Multiconference – ESM’2001*, Prague, Czech Republic, June 6-9 2001.
- (Vee & Hsu, 1999) VEE, V.; HSU, W. Parallel Discrete Event Simulation: A Survey, *Technical Report* at Centre for Advanced Information Systems (CAIS) – Nanyang Technological University, 1999.
- (Wagner & Lazowska, 1989) WAGNER, D. B.; LAZOWSKA, E. D. Parallel Simulation of Queueing Networks: Limitations and Potentials In: *Performance Evaluation Review*, v. 17, n. 1, May 1989.
- (Weiss & Kulikowski, 1991) WEISS, S. M.; KULIKOWSKI C. A. *Computer Systems That Learn* Morgan Kaufmann Publishers, Inc., California, 1991.
- (Wieland, 2001) WIELAND, F. Pratical Parallel Simulation Applied to Aviation Modeling. In: *Proceeding of the 15th Workshop on Parallel and Distributed Simulation (PADS’2001)*, California - USA, 2001.
- (Wildt & Wagner, 2001) WILDT, D. F., WAGNER, F. R. Adapting Simulation Environments to HLA: Discussion and Case Study. In: *Proceedings of the European Simulation Multiconference ESM2001*, Prague, Czech Republic, June 2001.
- (Wilson & Weatherly, 1994) WILSON, A.; Weatherly, R. The Aggregate Level Simulation Protocol: An Evolving System. In: *Proceedings of the 1994 Winter Simulation Conference*, Florida - USA, December 1994.
- (Yau & Pawlikowski, 2000) YAU, V., PAWLIKOWSKI, K. A Parallel Simulation Methodology for Speedup and Obtaining Performance Estimates with Specific Accuracy: Experiences of its Application in Studies of Metropolitan Area Networks. In: *Simulation*, v. 75, n. 4, p. 211-224, 2000.

Apêndice A – Computação Paralela / Distribuída

Desde que surgiu, há 20 anos atrás, a Computação Paralela vem permitindo que problemas complexos sejam solucionados e aplicações de alto desempenho sejam desenvolvidas tanto nas áreas mais tradicionais, tais como ciência e engenharia, como nas áreas novas, tais como inteligência artificial (*data mining*, redes neurais, etc.), financeira e bioinformática (Skillicorn & Talia, 1998).

Contudo, introduzir o conceito processamento paralelo não é tão simples. Existem diversas formas de organizar os processadores em paralelo além da necessidade de manter as informações que trafegam pela máquina coerentes e de gerenciar os elementos de processamento (processadores). O tempo de execução de um programa seqüencial não varia muito quando a execução muda de uma máquina seqüencial para outra. Isto já não pode ser observado quando se trata de programas paralelos, pois um fator muito importante na elaboração de programas paralelos é a comunicação entre os elementos de processamento e esta varia muito de um sistema para outro (Skillicorn & Talia, 1998).

Existem diferentes propostas para a classificação de arquiteturas de computadores paralelas, no entanto, devido à sua constante evolução, nenhuma classificação consegue abranger todas as existentes.

A classificação de Flynn (Flynn, 1972) é a mais conhecida e utilizada embora seja antiga e não consiga englobar todos os tipos de arquiteturas existentes atualmente. A classificação de Flynn baseia-se na unicidade e multiplicidade do fluxo de instruções e dados. Duncan (Duncan, 1990) propôs uma classificação mais recente e abrange a maior parte das arquiteturas, porém é menos conhecida.

Dentre as arquiteturas classificadas por Flynn (1972) e por Duncan (1990) destaca-se, para as execuções das simulações, a utilização dos multiprocessadores com memória compartilhada e dos multicomputadores com memória distribuída. O que distingue essas arquiteturas é o modo de acesso à memória.

A estrutura dos multiprocessadores com memória compartilhada pode ser observada na figura A.1. Os multiprocessadores são compostos por vários processadores conectados à memória por um barramento de alta velocidade. Normalmente, os dados ficam armazenados nos *caches* locais e um mecanismo de hardware se encarrega de mover os dados para a memória principal. A comunicação entre os processadores é feita através de variáveis compartilhadas e alguns protocolos de consistência devem garantir que essas variáveis estejam sempre atualizadas e coerentes. Os sistemas Sun Enterprise e Silicon Graphics Origin são exemplos de multiprocessadores com memória compartilhada (Fujimoto, 2000).

Figura A.1. Estrutura de um multiprocessador com memória compartilhada

A estrutura de um multicomputador de memória distribuída pode ser observada na figura A.2. Os multicomputadores de memória distribuída possuem vários nós, cada um com um processador e memória local, sendo estes conectados por uma rede de alta velocidade. A comunicação entre os processadores é feita através de trocas de mensagens. Os controladores de comunicação são responsáveis pelo envio e recebimento de mensagens. Os sistemas Cray T3D, NCube/Ten e Intel Paragon são exemplos de multicomputadores com memória distribuída (Fujimoto, 2000).

Figura A.2. Estrutura de um multicomputador com memória distribuída

Pode-se dizer que atualmente, além dos computadores paralelos, os computadores distribuídos também têm sido bastante utilizados para as execuções das simulações, baseando-se no conceito de computação paralela sobre sistemas distribuídos (Souza, 1996). O que distingue os computadores paralelos dos distribuídos são a heterogeneidade das máquinas e a rede de comunicação usada na conexão dos computadores. Enquanto nos computadores paralelos os nós são todos homogêneos e as redes são dedicadas e muito rápidas, nos computadores distribuídos os nós são heterogêneos (normalmente computadores pessoais ou estações de trabalho) e a comunicação é feita através de redes Ethernet ou ATM (Fujimoto, 2000).

A computação paralela distribuída tem por objetivo fazer uso dos computadores disponíveis nos sistemas distribuídos e transformá-los em unidades de processamento de um computador paralelo. Isso se torna possível através da utilização de ambientes de passagem de mensagem, também conhecidos por ambientes paralelos virtuais, os quais são descritos a seguir.

A.1. Ambientes de Passagem de Mensagem

Os ambientes de passagem de mensagem foram desenvolvidos inicialmente para processamento maciçamente paralelo (*Massively Parallel Processing* - MPP). Exemplos desses sistemas são: PSE da *nCUBE*, *EUI* da *IBM*, *CS System* da *Meiko* e *CMMD* da *Thinking Machines*. O problema desses ambientes é que, apesar de fornecerem serviços semelhantes (sincronização e troca de dados), foram implementados em sistemas totalmente incompatíveis pois cada fabricante desenvolvia o seu próprio ambiente (McBryan, 1994).

Atualmente, os ambientes de passagem de mensagens foram remodelados e/ou desenvolvidos com três grandes objetivos (Souza, 1996):

1. Utilizar o potencial dos sistemas distribuídos para o desenvolvimento de aplicações paralelas;
2. Permitir a união de plataformas heterogêneas;
3. Permitir a portabilidade das aplicações paralelas desenvolvidas.

Baseados nesses objetivos e também devido a diversidade encontrada nos ambientes já existentes, vários grupos foram formados com a finalidade de desenvolver as **plataformas de portabilidade de passagem de mensagem**. Essas plataformas têm por objetivo tornar os ambientes portáteis, ou seja, os programas podem ser executados em qualquer equipamento para o qual o ambiente foi desenvolvido. Exemplos dessas plataformas são: P4, PARMACS, EXPRESS, Zipcode, PVM, MPI e Linda (McBryan, 1994).

Na próxima subseção o ambiente de portabilidade PVM (Geist *et al.*, 1994) será descrito com maiores detalhes por ser uma ferramenta utilizada no desenvolvimento deste projeto.

A.2. PVM – Parallel Virtual Machine

PVM (*Parallel Virtual Machine*) é um conjunto integrado de bibliotecas de funções e de ferramentas de software, cuja finalidade é emular um sistema computacional concorrente heterogêneo, flexível e de propósito geral. Permite que uma coleção de sistemas de computadores heterogêneos possa ser visto como uma única máquina paralela virtual (Geist *et al.*, 1994). O projeto do PVM teve início em 1989 no *Oak Ridge National Laboratory* - ORNL e seu protótipo (versão 1.0) foi implementado por *Vaidy Sunderam* e *Al Geist* (ORNL). Com a versão 2.0 deu-se início a disponibilização do código. A versão mais recente

e disponível é a 3.4.4.

O sistema PVM permite que sejam escritas aplicações nas linguagens Fortran, C e C++. A escolha por esse conjunto de linguagens deve-se ao fato de que a maioria das aplicações passíveis de paralelização estão escritas nessas linguagens (Geist *et al.*, 1994).

Como vantagens da utilização do PVM pode-se citar: o ambiente é amplamente utilizado tanto pela comunidade científica quanto comercial, ou seja, é de grande aceitação; pode-se fazer utilização da computação paralela em equipamentos não paralelos, reduzindo o tempo de processamento; o software é de domínio público, flexível e também de fácil programação, pois é uma extensão de linguagens conhecidas.

A.2.1. O modelo computacional do PVM

No PVM, um conjunto de computadores seriais, paralelos e vetoriais emula um grande computador de memória distribuída (denominada *máquina virtual*). Cada componentes desta máquina virtual será denominado *host*. Vários usuários podem definir diferentes máquinas virtuais na mesma rede, podendo algumas máquinas fazer parte de mais de uma máquina virtual, executando aplicações PVM simultaneamente. PVM fornece funções para o início das tarefas (*tasks*) na máquina virtual e também funções para que estas tarefas se comuniquem e sincronizem entre si. Uma tarefa é definida como uma unidade de computação, semelhante à um processo UNIX (Geist *et al.*, 1994; Sunderam *et al.*, 1994).

Uma aplicação é composta por diversas tarefas segundo o modelo computacional do PVM, assumindo-se que qualquer tarefa pode enviar uma mensagem para uma qualquer outra tarefa. Cada tarefa é responsável por uma parte da carga de trabalho computacional da aplicação. Algumas vezes a aplicação pode ser paralelizada segundo suas funções, ou seja, cada tarefa realiza uma função diferente, como, por exemplo, entrada das informações, solução e exibição dos resultados. Esta forma de paralelizar é denominada *paralelismo funcional*. Outra forma de paralelizar uma aplicação é denominada *paralelismo de dados*. Nesta forma, todas as tarefas são as mesmas, mas cada uma soluciona somente uma parte da aplicação, dependendo dos dados que lhe são fornecidos (Geist *et al.*, 1994). O paralelismo de dados refere-se ao paradigma SPMD (Single Program, Multiple Data). Neste paradigma, um único código fonte é executado em diversos processadores independentemente uns dos outros. Quando diversos códigos fontes são distribuídos pelos processadores, tem-se o paradigma MPMD (Multiple Program, Multiple Data).

Em termos de implementação, o modelo computacional do PVM é composto por duas partes: um *daemon* (usualmente denotado por *pvmd*) e uma biblioteca de rotinas com a interface PVM (usualmente denotada por *libpvm*).

O usuário que deseje utilizar o PVM deve primeiramente configurar a máquina virtual especificando a lista de *hosts* e iniciar o *daemon* em todas as máquinas que fazem parte da

máquina virtual. Normalmente este programa é executado em *background*, ou seja, o usuário não percebe que ele está sendo executado. O primeiro *pvmd* (iniciado pelo usuário) é denominado mestre, enquanto que os outros que são criados pelo mestre são denominados escravos. A diferença é que somente o mestre pode iniciar outros *pvmds* e adicioná-los à máquina virtual (Souza, 1996). O *pvmd* não faz nenhum processamento, somente roteamento e controle de mensagens, ou seja, funciona como um ponto de contato entre os *hosts* autenticando as mensagens, fazendo o controle de processos e detectando possíveis falhas.

Na *libpvm* encontram-se as primitivas necessárias à cooperação entre as tarefas de uma aplicação, funcionando como um elo de ligação entre uma tarefa e a máquina virtual, ou seja, são rotinas que podem ser chamadas pelo usuário para efetuar troca de mensagens, solicitar a geração de processos, coordenar tarefas e solicitar modificações na máquina virtual. É uma biblioteca pequena tendo em vista que compartilha o espaço de endereçamento com o código do usuário. (Geist *et al.*, 1994).

A.2.2. Comunicação e Sincronismo no PVM

Como já comentado, o PVM oferece rotinas para o envio de mensagens entre as tarefas que fazem parte de uma aplicação. Não há limites no tamanho ou número dessas mensagens (Geist *et al.*, 1994).

O envio de uma mensagem utilizando o PVM é composto por três etapas:

- Criação de um *buffer* para envio;
- Empacotamento da mensagem neste *buffer*;
- Envio da mensagem para outra tarefa.

O recebimento da mensagem é feito através da chamada de uma função *receive*, que pode ser bloqueante ou não bloqueante, seguido pelo desempacotamento da mensagem (Geist *et al.*, 1994).

O mecanismo de envio é denominado *send bloqueante assíncrono*. É bloqueante pois o *send* bloqueia o processo quando o tamanho da mensagem excede o tamanho do *buffer* de envio e a mensagem precisa ser dividida. Neste caso é necessário esperar o recebimento da mensagem pela outra tarefa (através de uma *receive*) para que o *buffer* seja esvaziado e o resto da mensagem possa ser enviado. Caso a mensagem não exceda o tamanho do *buffer* de envio, o *send* retorna tão logo o *buffer* de envio esteja vazio. É assíncrono pois não depende do receptor executar um *receive* para poder retornar.

Com o *receive* bloqueante a execução da tarefa continua somente quando existem dados no *buffer* de recebimento. Já com o *receive* não bloqueante, a execução da tarefa irá continuar imediatamente, independente da haver dados no *buffer* de recebimento ou não. Neste caso uma variável (*flag*) é retornada indicando se há ou não dados.

O PVM garante que a ordem das mensagens é preservada, isto é, as mensagens enviadas de uma tarefa para outra chegarão na mesma ordem em que foram enviadas.

Além da comunicação ponto-a-ponto, o PVM oferece comunicação *broadcasting* (envio de uma mensagem para um grupo de tarefas) e *multicasting* (envio de uma mensagem para um conjunto de tarefas).

Para a comunicação, o PVM utiliza os protocolos TCP (*Transmission Control Protocol*), UDP (*User Datagram Protocol*) e *sockets* do domínio UNIX. Três formas de comunicação podem ser estabelecidas: entre os *pvmd* (através do protocolo UDP), entre o *pvmd* e as tarefas e entre as tarefas (através do protocolo TCP e de *sockets*) (Geist *et al.*, 1994).

Referências:

- (Duncan, 1990) DUNCAN, R. A survey of parallel computer architectures. *IEEE Computer*. v.23, n.2, p.5-16, 1990.
- (Flynn, 1972) FLYNN, M. J. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, v.C-21, p.948-960, 1972.
- (Geist *et al.*, 1994) GEIST, A.; BEGUELIN, A.; DONGARRA, J.; JIANG, W.; MANCHEK, R.; SUNDERAM, V. *PVM: Parallel Virtual Machine – A Users' Guide and Tutorial for Networked Parallel Computing*, Cambridge: The MIT Press, 1994
- (McBryan, 1994) MCBRYAN, O.A. An Overview of Message Passing Environments. *Parallel Computing*, v.20, p.417-444, 1994.
- (Skillicorn & Talia, 1998) SKILLICORN, D.B.; TALIA, D. Models and Languages for Parallel Computation. *ACM Computing Surveys*, v.30, n.2, 1998, pp 123-169.
- (Souza, 1996) SOUZA, P.S.L. Máquina Paralela Virtual em Ambiente Windows. São Carlos, 1996. Dissertação (Mestrado) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.

Apêndice B – UML (*Unified Modelling Language*)

O desenvolvimento da UML teve início em 1994 com o objetivo de tornar um padrão internacional de linguagem para modelagem de sistemas orientados a objetos. Esta linguagem pretende substituir todos os métodos de projeto e análise de software (tais como Booch, Coad, Jacobson, Odell, Rumbaugh, Wirfs-Brock entre outros) com uma única notação, apoiados pela OMG (*Object Management Group*) (Kobryn, 1999; Ambler, 2000).

A UML é uma linguagem gráfica para visualização, especificação, construção e documentação de artefatos de sistemas complexos de *software* (Booch *et al.*, 2000), propondo nove diagramas e um conjunto de elementos que podem ser utilizados em quaisquer desses diagramas. Os diagramas fornecem suporte para diferentes fases de desenvolvimento (desde a especificação dos requisitos até a implementação) e também diferentes visões e níveis de abstração do sistema.

A arquitetura de sistema pode ter 5 visões complementares e inter-relacionadas, como ilustrado na figura B.1 (Booch *et al.*, 2000). Em conjunto, essas visões representam a base do projeto do software e podem conter aspectos estruturais (características estáticas) e comportamentais (características dinâmicas).

Figura B.1. Modelagem da arquitetura de um sistema

A visão de caso de uso abrange os casos de uso que descrevem o comportamento do sistema conforme é isto pelos seus usuários finais. Os aspectos estáticos dessa visão são descritos através dos diagramas de caso de uso e os aspectos dinâmicos através dos diagramas de interação (seqüência e colaboração), gráficos de estado e atividades.

A visão de projeto abrange as classes, interfaces e colaborações que formam o vocabulário do problema e sua solução. Esta visão proporciona suporte para os requisitos funcionais do sistema, ou seja, os serviços que o sistema deverá fornecer aos usuários finais. Os aspectos estáticos dessa visão são descritos através dos digramas de classes e objetos e os

aspectos dinâmicos através dos diagramas de interação (seqüência e colaboração), diagramas de estado e atividade.

A visão do processo abrange os threads e os processos que formam os mecanismos de concorrência e de sincronização do sistema. Os aspectos estáticos e dinâmicos dessa visão são descritos nos mesmos tipos de diagramas de visão de projeto, mas com o foco voltado para as classes ativas que representam esses threads e processos.

A visão de implementação abrange os componentes e o arquivos utilizados para a montagem e fornecimento do sistema físico. Os aspectos estáticos são descritos através dos diagramas de componentes e os aspectos dinâmicos através de diagramas de interação (seqüência e colaboração), diagramas de estado e atividade.

A visão de implantação abrange os nós que formam a topologia de hardware em que o sistema é executado. Essa visão direciona principalmente a distribuição, fornecimento e instalação das partes que constituem o sistema físico. Os aspectos estáticos são descritos nos diagramas de implantação e os aspectos dinâmicos através de diagramas de interação (seqüência e colaboração), diagramas de estado e atividade.

Dependendo da natureza do sistema uma ou mais visões podem ser focadas. Por exemplo, em sistemas que utilizam muitos dados predominarão modelos voltados para a visão estática do sistema. Em sistemas centrados na interface gráfica (uso de GUI), as visões estáticas e dinâmicas dos casos de uso são mais importantes. Em sistemas de tempo real a visão mais importante é a dinâmica. Já em sistemas distribuídos (aplicações Web), os modelos de implantação e implementação são os mais importantes.

A UML pode ser compreendida a partir de três elementos principais: blocos de construção básicos da UML, regras que determinam como esses blocos podem ser combinados e alguns mecanismos básicos aplicados na UML (Booch *et al.*, 2000). A seguir esses blocos são brevemente descritos. Maiores detalhes podem ser encontrados em (Booch *et al.*, 2000).

B.1 Blocos de construção da UML

O vocabulário da UML abrange três blocos de construção:

- Itens;
- Relacionamentos;
- Diagramas.

B.1.1 Itens

Os itens constituem os blocos básicos orientados a objetos e são quatro tipos:

- Estruturais;
- Comportamentais;
- Agrupamento;
- Anotacionais.

B.1.1.1 Itens Estruturais

Os itens estruturais são a parte mais estática do modelo e representam elementos conceituais ou físicos. São considerados os substantivos do modelo. Podem ser: classes, interfaces, colaborações, casos de uso, classes ativas, componentes ou nós.

As *classes* são os blocos de construção mais importantes de qualquer sistema orientado a objetos. Uma classe é uma descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica. As classes podem incluir abstrações que são parte do domínio do problema e podem ser utilizadas para representar itens de software, de hardware e até itens puramente conceituais. Graficamente as classes são representadas por retângulos que incluem o nome, atributos e operações (figura B.2). Essa representação permite visualizar a abstração independente de qualquer linguagem de programação.

Figura B.2. Representação gráfica de uma classe

A *interface* é uma coleção de operações que especificam serviços de uma classe ou componente, descrevendo o comportamento externamente visível desse elemento. Uma interface poderá representar todo ou apenas parte do comportamento. Graficamente é representada por um círculo e o respectivo nome (figura B.3) e está sempre associada à classe ou componente que realiza a interface.

Figura B.3. Representação gráfica de uma interface

As *colaborações* definem interações e são sociedades de papéis e outros elementos que funcionam em conjunto para proporcionar um comportamento cooperativo superior à soma de todos os elementos. Uma classe poderá participar de várias colaborações. Graficamente, as colaborações são representadas através de elipses com linhas tracejadas, incluindo seu nome (figura B.4)

Figura B.4. Representação gráfica de uma colaboração

Um *caso de uso* é a descrição de um conjunto de seqüências de ações realizadas pelo sistema que proporciona resultados observáveis de valor para um determinado ator. Graficamente, um caso de uso é representado por uma elipse com linhas contínuas, incluindo seu nome (figura B.5).

Figura B.5. Representação gráfica de um caso de uso

As *classes ativas*, *componentes* e *nós* são semelhantes às classes (compartilham os mesmos atributos e operações) porém possuem características suficientemente diferentes e necessárias a modelagem.

As classes ativas são classes cujos objetos ativos, ou seja, são objetos que possuem processos ou threads que são capazes de iniciar uma atividade de controle. Graficamente elas são representadas da mesma maneira que as classes com linhas mais grossas (figura B.6).

Figura B.6. Representação gráfica de uma classe ativa

Os componentes são partes físicas e substituíveis de um sistema, que proporciona a realização de um conjunto de interfaces. Tipicamente eles representam o pacote físico de elementos lógicos diferentes, como classes, interfaces e colaborações. Graficamente são representados por retângulos com abas, incluindo somente seus nomes (figura B.7).

Figura B.7. Representação gráfica de um componente

Os nós são elementos físicos existentes em tempo de execução que representam um recurso computacional, geralmente com pelo menos alguma memória e, freqüentemente, com capacidade de processamento. Graficamente, um nó é representado como um cubo, incluindo seu nome (figura B.8).

Figura B.8. Representação gráfica de um nó

B.1.1.2. Itens Comportamentais

Os itens comportamentais são as partes dinâmicas do modelo de UML. São considerados os verbos do modelo, representando comportamento no tempo e no espaço. Existem dois tipos de itens comportamentais: interação e máquina de estado.

A *interação* é um comportamento que abrange um conjunto de mensagens trocadas entre um conjunto de objetos em determinado contexto para a realização de propósitos específicos. As interações envolvem outros elementos, incluindo mensagens, seqüências de ações (os comportamentos chamados pelas mensagens) e ligações (as conexões entre os objetos). Graficamente, uma mensagem é representada como uma linha cheia com seta, incluindo o nome da operação (figura B.9).

Figura B.9. Representação gráfica de uma mensagem

A *máquina de estado* é um comportamento que especifica as seqüências de estados pelas quais objetos ou interações passam durante sua existência em resposta a eventos, bem como suas respostas a esses eventos. Uma máquina de estado abrange outros elementos, incluindo estados, transições (fluxo de um estado a outros), eventos (itens que disparam uma transição) e atividades (respostas às transições). Graficamente, o estado é representado como retângulo com ângulos arredondados, incluindo o nome e subestados (figura B.10).

Aguardando

Figura B.10. Representação gráfica de um estado

B.1.1.3 Itens Agrupamento

Os itens de agrupamento são as partes organizacionais dos modelos de UML. São os blocos em que os modelos podem ser decompostos. Existe somente um tipo, o pacote.

Os *pacotes* são mecanismos de propósito geral para a organização dos elementos em grupos. Os itens estruturais, organizacionais e outros itens podem ser colocados em pacotes. Ao contrário dos componentes, que existem em tempo de execução, os pacotes são puramente conceituais. Graficamente são representados como diretórios com guias, incluindo o nome e às vezes, o conteúdo (figura B.11)

Figura B.11. Representação gráfica de um pacote

B.1.1.4 Itens Anotacionais

Os itens anotacionais são as partes explicativas dos modelos de UML. São comentários, incluídos para descrever, esclarecer e fazer alguma observação sobre qualquer elemento do modelo. Existe somente um tipo denominado nota e é utilizada para aprimorar os diagramas com restrições ou comentários que possam ser mais bem expressos por um texto formal ou informal. É representada graficamente por um retângulo com um dos cantos como uma dobra de página, acompanhada por texto ou comentários gráficos (figura B.12).

Figura B.12. Representação gráfica de uma nota

B.1.2 Relacionamentos

Os relacionamentos são os blocos relacionais básicos de construção da UML e são

utilizados para escrever modelos bem-formados. Existem quatro tipos:

- Dependência;
- Associação;
- Generalização;
- Realização

B.1.2.1 Dependência

Uma dependência é um relacionamento semântico entre dois itens, nos quais a alteração de um (item independente) pode afetar a semântica do outro (item dependente). Gráficamente é representada por linhas tracejadas, possivelmente com setas e ocasionalmente incluindo um rótulo (figura B.13).

Figura B.13. Dependência

B.1.2.2 Associação

A associação é um relacionamento estrutural que descreve um conjunto de ligações, em que essas ligações são conexões entre objetos. Gráficamente é representada por linhas sólidas contendo outros adornos, tais como nomes de papéis e multiplicidade (figura B.14).

Figura B.14. Associação

B.1.2.3 Generalização

A generalização é um relacionamento de especialização/generalização onde os objetos dos elementos especializados (filhos) são substituíveis por objetos do elemento generalizado (pais). Gráficamente a generalização é representado como linha sólida com uma seta em branco apontando o pai (figura B.15).

Figura B.14. Generalização

B.1.2.4. Realização

Uma realização é um relacionamento semântico entre classificadores, onde um classificador classifica um contrato que outro classificador garante executar. Podem ser encontrados em dois locais: entre interfaces e as classes ou componentes que as realizam e entre casos de uso e as colaborações que os realizam. Graficamente é representado por uma linha tracejada com seta branca entre uma generalização e um relacionamento de dependência (figura B.15).

Figura B.15. Realização

B.1.3 Diagramas

Um diagrama é uma representação gráfica de um conjunto de uma coleção de itens e relacionamentos onde, freqüentemente, os arcos são os relacionamentos e os vértices são os itens. Os diagramas podem ser divididos em dois tipos: estruturais e comportamentais.

Os diagramas estruturais servem para visualizar, especificar, construir e documentar os aspectos estáticos do sistema, ou seja, aqueles onde a estrutura descrita é sempre válida em qualquer ponto do ciclo de vida do sistema. São eles (Booch *et al.*, 2000):

- Diagrama de classes, composto por classes, interfaces, pacotes, colaborações e relacionamentos de dependência, generalização e associação;
- Diagrama de objetos, composto por objetos e vínculos;
- Diagrama de componentes, composto por componentes, interfaces, colaborações e relacionamentos de dependência, generalização, associação e realização;
- Diagrama de implantação, composto por nós e relacionamentos de dependência e associação

Os diagramas comportamentais servem para visualizar, especificar, construir e documentar os aspectos dinâmicos do sistema. São eles:

- Diagrama de casos de uso, composto por casos de uso, atores e relacionamentos de dependência, generalização e associação;
- Diagrama de interação (seqüências e colaborações), composto por objetos, vínculos e mensagens;
- Diagrama de atividades, composto por estados de atividade e estados de ação, transições e objetos;

- Diagrama de estado, composto por estados (simples e compostos) e transições (incluindo eventos e ações)

A seguir cada um dos diagramas é definido e exemplificado.

B.1.3.1 Diagramas de classes e objetos

Os diagramas de classes representam as classes (incluindo os atributos e operações), interfaces, pacotes e colaborações que farão parte do sistema e também o relacionamento entre elas. Neste diagrama, as classes são documentadas com a descrição do que elas fazem, as operações com a descrição da lógica e os atributos através da descrição do que eles contêm, do tipo e da possível variação.

O diagrama de objetos utiliza a mesma notação do digrama de classes mas, ao invés de apresentar as classes, apresenta as instâncias das classes (objetos), com o nome do objeto sublinhado dentro do retângulo da classe.

Um exemplo de um diagrama de classe pode ser observado na figura B.16.

Figura B.16. Diagrama de classes

B.1.3.2 Diagramas de casos de uso

O diagrama casos de uso é usado para identificar o comportamento do sistema em várias situações, descrevendo a interação entre o sistema e o mundo externo. É composto por dois

componentes: o ator e os casos de uso.

O ator representa as entidades que interagem com o sistema, mas são externas a ele. Os casos de uso são seqüências de ações que o sistema executa e produzem um resultado para o ator.

Os casos de uso descrevem *quem* (ator) faz *o que* (interação) com o sistema (ou um subsistema) em relação a um determinado propósito (objetivo), mas não especifica *como* isso é feito. Ao fazer a modelagem é importante manter clara a separação entre as visões interna e externa.

O diagrama de casos de uso é acompanhado por uma documentação e pela definição de cenários. A documentação é composta por uma descrição textual resumida e pelos fluxos de eventos (fluxo principal, subfluxos alternativos e subfluxos de execução). Os cenários são instâncias de casos de uso e devem ser construídos em número suficiente para que seja possível o entendimento completo do sistema.

A forma de um diagrama de casos de uso é apresentada na figura B.17.

Figura B.17. Diagrama de Casos de Uso.

B.1.3.3. Diagramas de seqüências

Este diagrama é utilizado para definir rigorosamente a lógica dos cenários de casos de uso. Ele mostra a interação entre os objetos envolvidos nos casos de uso, as mensagens que são trocadas entre si e os valores retornados. As duas dimensões de um diagrama de seqüência consistem na dimensão vertical (tempo) e na dimensão horizontal (objetos diferentes). A forma de um diagrama de seqüências pode ser observada na figura B.18.

B.1.3.4. Diagrama de colaborações

O diagrama de colaboração estende o diagrama de seqüência mostrando os relacionamentos entre os objetos. Ele mostra o fluxo de mensagens utilizando números nas seqüências de forma a evidenciar a seqüência de mensagens entre os objetos. Este diagrama não considera as linhas de tempo. O exemplo de um diagrama de colaborações é apresentado na figura B.19.

Figura B.18. Diagrama de Seqüência

Figura B.19. Diagrama de Colaboração

B.1.3.5. Diagramas de estados

Os objetos possuem comportamento e estado. Alguns objetos são tão complexos que é necessário um diagrama de estados para descrever como eles trabalham. Este diagrama representa a seqüência de estados que faz com que um objeto passe de um estado para outro em resposta a estímulos recebidos. O exemplo de um diagrama de estado pode ser observado na figura B.20

Figura B.20. Diagrama de Estados

B.1.3.6. Diagramas de atividades

O diagrama de atividades define uma visão estendida do diagrama de estados. Todos (ou pelo menos todos) os estados são ações e todas (ou pelo menos todas) as transições são disparadas pela finalização das ações nos estados de origem. Este diagrama é dirigido ao fluxo do processamento interno. A figura B.21 ilustra este diagrama.

Figura B.21. Diagrama de Atividades

B.1.3.7. Diagramas de componentes

O diagrama de componentes mostra as dependências entre os componentes de software incluindo códigos fonte, binário e executável. Alguns dos componentes existem somente no

momento da compilação, outros já existem somente quando o sistema está sendo executado, enquanto outros podem existir em vários momentos ao mesmo tempo. Um exemplo de um diagrama de componentes é ilustrado na figura B.22.

Figura B.22. Diagrama de Componentes

B.1.3.8. Diagramas de implantação

O diagrama de implantação mostra a configuração dos elementos de processamento em tempo de execução e os componentes de software, processos e objetos que pertencem ao elemento de processamento. A figura B.23 ilustra um diagrama de implantação.

Figura B.23. Diagrama de Implantação

Referências:

- (Ambler, 2000) AMBLER S. W. How the UML Models fit together. *Software Development's* special report on the UML, March, 2000
- (Booch *et al.*, 2000) BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML: Guia do Usuário* Editora Campus, 1^a edição, 2000.
- (Fowler, 2000) FOWLER, M. Why use the UML? *Software Development's* special report on the UML, March, 2000
- (Kobryn, 1999) KOBRYN, C. UML 2001: A Standardization Odyssey. *Communications of the ACM*. v. 42, n. 10. October 1999.

Apêndice C – Códigos dos Programas de Simulação

```
#include <stdio.h>
#include <sys/time.h>
#include "smp1x.h"
#include "distributions.h"    // para as distribuicoes
#include "debug.h"           // para os arquivos de debug
#include "communication.h"    // para a comunicacao

main()
{
    /* definicoes */
    float Te = 100000;
    int Event = 1, Customer = 1;
    double Tal = 0.0833, Tsl = 0.05;
    double busy;
    int metodo_analise;
    int id, parent, seed;

    id = GetID();
    parent = GetParent();
    if (Open_Data_File(id) == 0)
    {
        DUMP_TEXT("Erro na abertura do arquivo de dados");
        Close_Debug_File();
        SendInt(parent, M_EAE);
        exit(0);
    }

    if (Open_Debug_File(id) == 0)
    {
        DUMP_TEXT("Erro na abertura do arquivo de debug");
        Close_Data_File();
        Close_Debug_File();
        SendInt(parent, M_EAE);
        exit(0);
    }

    metodo_analise = start_communication();
    if (metodo_analise == -1)
    {
        DUMP_TEXT("Erro no inicio da comunicacao com o mestre");
        Close_Debug_File();
        Close_Data_File();
        SendInt(parent, M_ECE);
        exit(0);
    }

    Inicializa_MetodoAnalise(metodo_analise);

    ReceiveInt(parent, &seed);
    Inicializa_CMRRandomGenerator((unsigned long)seed);

    /* prepara o sistema de simulacao e da nome ao modelo */
    smp1(0, "teste");

    /* cria e da nome as facilidades */
    facility("teste", 1);
}
```

```

/* escalona a chegada do primeiro cliente */
schedule(1,0.0, Customer);

while(1)
{
    cause(&Event, &Customer);
    switch(Event)
    {
        case 1:
            schedule(2,0.0, Customer);
            schedule(1, Exponential(Ta1), Customer);
            break;
        case 2:
            if (request("teste", 2, Customer, 0) == 0)
                schedule(3, Exponential(Ts1), Customer);
            break;
        case 3:
            release("teste", Customer);
            busy = utilizacao_recurso("teste");
            ProcessObservation(busy);
            break;
    }
}

Close_Debug_File();
Close_Data_File();

} // programa principal

```

Figura C.1. Código MRIP gerado a partir de um programa de simulação seqüencial utilizando a biblioteca SMPL.

```

/ -----( MM1.CPP )-----
#include "queuing.h"
#include "distributions.h"    // para as distribuicoes
#include "debug.h"           // para os arquivos de debug
#include "communication.h"   // para a comunicacao

const int    TEMPO_SIMUL = 1000;
const double MEDIA_CHEGADA = 0.0833
const double MEDIA_SERVICO = 0.05;
int metodo_analise;
int id, parent, seed;

enum TipoEvento {CHEGADA, REQ_CPU, LIBERA_CPU};

void Chegada      (void);
void ReqCpu       (void);
void LiberaCpu    (void);

Facility *cpu     = NULL;

Token cliente;

int main()
{
    id = GetID();
    parent = GetParent();
    if (Open_Data_File(id) == 0)

```

```

{
    DUMP_TEXT("Erro na abertura do arquivo de dados");
    Close_Debug_File();
    SendInt (parent,M_EAE);
    exit(0);
}

if (Open_Debug_File(id) == 0)
{
    DUMP_TEXT("Erro na abertura do arquivo de debug");
    Close_Data_File();
    Close_Debug_File();
    SendInt (parent,M_EAE);
    exit(0);
}

metodo_analise = start_communication();
if (método_analise == -1)
{
    DUMP_TEXT("Erro no inicio da comunicacao com o mestre");
    Close_Debug_File();
    Close_Data_File();
    SendInt (parent,M_ECE);
    exit(0);
}

Inicializa_MetodoAnalise(metodo_analise);

ReceiveInt (parent,&seed);
Inicializa_CMRRandomGenerator((unsigned long)seed);

new Future(LINKED);
cpu = new Facility("CPU", 1);

cliente.Id(1);
Future::Schedule(CHEGADA, 0.0, cliente);

while (Future::SimTime() < TEMPO_SIMUL)
{
    Estatus es = Future::NextEvent();
    cliente = es.token;
    switch(es.event_id)
    {
        case CHEGADA      : Chegada();      break;
        case REQ_CPU      : ReqCpu();        break;
        case LIBERA_CPU   : LiberaCpu();     break;
        default           : ErrXit (1, "event_id invalido");
    }
}

Close_Debug_File();
Close_Data_File();

Future::ReportStats();
} // end main

void Chegada()
{
    double amostra = Exponential(MEDIA_CHEGADA);
    Future::UpdateArrivals();
    Future::Schedule(REQ_CPU, 0.0, cliente);
}

```

```

// Escalona a chegada do proximo cliente
cliente.Id(cliente.Id() + 1);
Future::Schedule(CHEGADA, amostra, cliente);
}

void ReqCpu()
{
    if (cpu->Request(cliente) == FREE)
    {
        double amostra = Exponential(MEDIA_SERVICO);
        Future::Schedule(LIBERA_CPU, amostra, cliente);
        double t = cpu->Inuse_Time();
        ProcessObservation(busy);
    }
}

void LiberaCpu()
{
    int id = cliente.Id();
    cpu->Release(id);
    Future::Schedule(REQ_DISCO, 0.0, cliente);
}

```

Figura C.2. Código MRIP gerado a partir de um programa de simulação seqüencial utilizando a biblioteca SimPack.

```

#include <stdlib.h>
#include <stdio.h>
#include "sss.h"
#include "distributions.h"    // para as distribuicoes
#include "debug.h"           // para os arquivos de debug
#include "communication.h"   // para a comunicacao

char form[] = "%10.2f\n";

#define ARRIVL  1
#define STARTA  2
#define ENDACT  3

void main(void)
{
    int  event,      /* event code */
        seed,       /* random seed */
        busy = 0;   /* =1 if server busy */
    float periodo_ocupado, periodo_medio_ocupado;
    int count;
    int metodo_analise;
    int id, parent, seed;

    id = GetID();
    parent = GetParent();
    if (Open_Data_File(id) == 0)
    {
        DUMP_TEXT("Erro na abertura do arquivo de dados");
        Close_Debug_File();
        SendInt(parent, M_EAE);
        exit(0);
    }

    if (Open_Debug_File(id) == 0)
    {

```

```

    DUMP_TEXT("Erro na abertura do arquivo de debug");
    Close_Data_File();
    Close_Debug_File();
    SendInt(parent, M_EAE);
    exit(0);
}

metodo_analise = start_communication();
if (método_analise == -1)
{
    DUMP_TEXT("Erro no inicio da comunicacao com o mestre");
    Close_Debug_File();
    Close_Data_File();
    SendInt(parent, M_ECE);
    exit(0);
}

Inicializa_MetodoAnalise(metodo_analise);

ReceiveInt(parent, &seed);
Inicializa_CMRRandomGenerator((unsigned long)seed);

count = 0;
periodo_ocupado = periodo_medio_ocupado = 0;

INIQUE(1,1,1);
SETSEE(seed);
CREATE(EX(0.0833), 0);

do
{
    event = NEXTEV();
    if (event) switch(event)
    {
        case ARRIVL: SETA(1, T());
                     CREATE(Exponential(0.0833), IDE()+1);
                     if (busy) QUEUE(1,0);
                     else SCHED(0.0, STARTA, IDE());
                     break;

        case STARTA: busy = 1;
                     SCHED(Exponential(0.05), ENDACT, IDE());
                     break;

        case ENDACT: periodo_ocupado += (T() - A(1));
                     count++;
                     periodo_medio_ocupado = periodo_ocupado/count;
                     ProcessObservation(periodo_medio_ocupado);
                     DISPOS();
                     busy = 0;
                     if (NQ(1))
                     {
                         REMVFQ(1,1);
                         SCHED(0.0, STARTA, IDE());
                     }
                     break;
    }
} while (event);

Close_Data_File();
Close_Debug_File();

```

```
}

```

Figura C.3. Código MRIP gerado a partir de um programa de simulação seqüencial utilizando a biblioteca SSS

```
/* ----- */
#include <stdio.h>
#include <sys/time.h>
#include "smp1x.h"
#include "distributions.h" // para as distribuicoes
#include "debug.h" // para os arquivos de debug
#include "communication.h" // para a comunicacao

main()
{
    /* definicoes */
    float Te = 100000;
    int Event = 1, Customer = 1;
    double Tal = 2, Tsl = 1;
    double utilization;
    int metodo_analise;
    int id, parent, seed;

    id = GetID();
    parent = GetParent();
    if (Open_Data_File(id) == 0)
    {
        DUMP_TEXT("Erro na abertura do arquivo de dados");
        Close_Debug_File();
        SendInt(parent, M_EAE);
        exit(0);
    }

    if (Open_Debug_File(id) == 0)
    {
        DUMP_TEXT("Erro na abertura do arquivo de debug");
        Close_Debug_File();
        Close_Data_File();
        SendInt(parent, M_EAE);
        exit(0);
    }

    metodo_analise = start_communication();
    if (metodo_analise == -1)
    {
        DUMP_TEXT("Erro no inicio da comunicacao com o mestre");
        Close_Debug_File();
        Close_Data_File();
        SendInt(parent, M_ECE);
        exit(0);
    }

    parent = GetParent();
    ReceiveInt(parent, &seed);

    Inicializa_MetodoAnalise(metodo_analise);

    ReceiveInt(parent, &seed);
    Inicializa_CMRRandomGenerator((unsigned long) seed);
}
```



```

/* prepara o sistema de simulacao e da nome ao modelo */
smpl(0,"modelo M/M/1");

/* cria e da nome as facilidades */
facility("servidor",1);

/* escalona a chegada do primeiro cliente */
schedule(1,0.0,Customer);

contador = 0;

while(1)
{
    cause(&Event,&Customer);
    switch(Event)
    {
        case 1:
            schedule(2,0.0,Customer);
            schedule(1,Exponential(Ta1),Customer);
            break;
        case 2:
            if (request("servidor",2,Customer,0) == 0)
                schedule(3,Exponential(Ts1),Customer);
            break;
        case 3:
            release("servidor",Customer);
            busy = utilizacao_recurso("teste");
            ProcessObservation(busy);
            break;
    }
    Close_Debug_File();
    Close_Data_File();
}

```

Figura C.4. Código do modelo M/M/1

```

/* ----- */
#include <sys/time.h>
#include <stdio.h>
#include "smplx.h"
#include "distributions.h"    // para as distribuicoes
#include "debug.h"           // para os arquivos de debug
#include "communication.h"   // para a comunicacao

main()
{
    /* definicoes */
    int Event = 1, Customer = 1, Aleatorio, i;
    real Ts1 = 1.53686, Ts2 = 31.32981,
        Ts3 = 82.705, Ts4 = 135.7939,
        Ts5 = 136.6733;
    float utilization;
    int metodo_analise;
    int id, parent, seed;

    id = GetID();
    parent = GetParent();
    if (Open_Data_File(id) == 0)
    {
        DUMP_TEXT("Erro na abertura do arquivo de dados");
        Close_Debug_File();
    }
}

```

```

    SendInt (parent,M_EAE);
    exit (0);
}

if (Open_Debug_File(id) == 0)
{
    DUMP_TEXT("Erro na abertura do arquivo de debug");
    Close_Debug_File();
    Close_Data_File();
    SendInt (parent,M_EAE);
    exit(0);
}

metodo_analise = start_communication();
if (metodo_analise == -1)
{
    DUMP_TEXT("Erro no inicio da comunicacao com o mestre");
    Close_Debug_File();
    Close_Data_File();
    SendInt (parent,M_ECE);
    exit(0);
}

parent = GetParent();
ReceiveInt (parent,&seed);

Inicializa_MetodoAnalise(metodo_analise);

ReceiveInt (parent,&seed);
Inicializa_CMRRandomGenerator((unsigned long)seed);

/* prepara o sistema de simulacao e da nome ao modelo */
smp1(0," pvm");

/* cria e da nome as facilidades */
facility("pvm",1);
facility("lasdix",1);
facility("lasdpc08",1);
facility("lasdpc10",1);
facility("lasdpc11",1);

/* escalona a chegada do primeiro cliente */
for (i=0; i<=100; i++){
    schedule(1,0, Customer);}

while (1)
{
    cause(&Event,&Customer);
    switch(Event)
    {
        case 1:
            schedule(2,0.0,Customer);
            break;
        case 2:
            if (request("pvm",2,Customer,0) == 0)
                schedule(3,Exponential(Ts1),Customer);
            break;
        case 3:
            release("pvm", Customer);
            utilization = utilizacao_recurso("pvm");
            ProcessObservation(utilization);
            Aleatorio = UniformInt(1,10000);
            if ( (0 <= Aleatorio) && (Aleatorio <= 2500) )
                schedule(6, 0.0, Customer);
    }
}

```

```

        if ( (2501 <= Aleatorio) && (Aleatorio <= 5000) )
            schedule(8, 0.0, Customer);
        if ( (5001 <= Aleatorio) && (Aleatorio <= 7500) )
            schedule(10, 0.0, Customer);
        if ( (7501 <= Aleatorio) && (Aleatorio <= 10000) )
            schedule(4, 0.0, Customer);
        break;
    case 4:
        if (request("lasdix",4,Customer,0) == 0)
            schedule(5,Exponential(Ts2),Customer);
        break;
    case 5:
        release("lasdix", Customer);
        schedule(2, 0.0, Customer);
        break;
    case 6:
        if (request("lasdpc08",6,Customer,0) == 0)
            schedule(7,Exponential(Ts3),Customer);
        break;
    case 7:
        release("lasdpc08", Customer);
        schedule(2, 0.0, Customer);
        break;
    case 8:
        if (request("lasdpc10",8,Customer,0) == 0)
            schedule(9,Exponential(Ts4),Customer);
        break;
    case 9:
        release("lasdpc10", Customer);
        schedule(2, 0.0, Customer);
        break;
    case 10:
        if (request("lasdpc11",10,Customer,0) == 0)
            schedule(11,Exponential(Ts5),Customer);
        break;
    case 11:
        release("lasdpc11", Customer);
        schedule(2, 0.0, Customer);
        break;
    }

}

Close_Debug_File();
Close_Data_File();
}
/* ----- */

```

Figura C.5. Código do modelo do sistema distribuído

```

/* ----- */
#include <sys/time.h>
#include <stdio.h>
#include "smp1x.h"
#include "distributions.h" // para as distribuicoes
#include "debug.h"         // para os arquivos de debug
#include "communication.h" // para a comunicacao

main()
{
    /* definicoes */
    int Event = 1, Customer = 1, Aleatorio, i;
    real Ts1 = 1.53686, Ts2 = 31.32981,

```

```

        Ts3 = 82.705,   Ts4 = 135.7939,
        Ts5 = 136.6733;
float utilization;
int metodo_analise;
int id, parent, seed;

id = GetID();
parent = GetParent();
if (Open_Data_File(id) == 0)
{
    DUMP_TEXT("Erro na abertura do arquivo de dados");
    Close_Debug_File();
    SendInt(parent, M_EAE);
    exit(0);
}

if (Open_Debug_File(id) == 0)
{
    DUMP_TEXT("Erro na abertura do arquivo de debug");
    Close_Debug_File();
    Close_Data_File();
    SendInt(parent, M_EAE);
    exit(0);
}

metodo_analise = start_communication();
if (metodo_analise == -1)
{
    DUMP_TEXT("Erro no inicio da comunicacao com o mestre");
    Close_Debug_File();
    Close_Data_File();
    SendInt(parent, M_ECE);
    exit(0);
}

parent = GetParent();
ReceiveInt(parent, &seed);

Inicializa_MetodoAnalise(metodo_analise);

ReceiveInt(parent, &seed);
Inicializa_CMRRandomGenerator((unsigned long)seed);

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

/* cria e da nome as facilidades */
facility("pvm", 1);
facility("lasdix", 1);
facility("lasdpc08", 1);
facility("lasdpc10", 1);
facility("lasdpc11", 1);

/* escalona a chegada do primeiro cliente */
for (i=0; i<=100; i++){
    schedule(1, 0, Customer);}

while (1)
{
    cause(&Event, &Customer);
    switch(Event)
    {
        case 1:
            schedule(2, 0.0, Customer);
    }
}

```

```

        break;
    case 2:
        if (request("pvm",2,Customer,0) == 0)
            schedule(3,Exponential(Ts1),Customer);
        break;
    case 3:
        release("pvm", Customer);
        utilization = utilizacao_recurso("pvm");
        ProcessObservation(utilization);
        Aleatorio = UniformInt(1,10000);
        if ( (0 <= Aleatorio) && (Aleatorio <= 2500) )
            schedule(6, 0.0, Customer);
        if ( (2501 <= Aleatorio) && (Aleatorio <= 5000) )
            schedule(8, 0.0, Customer);
        if ( (5001 <= Aleatorio) && (Aleatorio <= 7500) )
            schedule(10, 0.0, Customer);
        if ( (7501 <= Aleatorio) && (Aleatorio <= 10000) )
            schedule(4, 0.0, Customer);
        break;
    case 4:
        for (i = 0; i<50000; i++) {}
        if (request("lasdix",4,Customer,0) == 0)
            schedule(5,Exponential(Ts2),Customer);
        break;
    case 5:
        release("lasdix", Customer);
        schedule(2, 0.0, Customer);
        break;
    case 6:
        if (request("lasdpc08",6,Customer,0) == 0)
            schedule(7,Exponential(Ts3),Customer);
        break;
    case 7:
        release("lasdpc08", Customer);
        schedule(2, 0.0, Customer);
        break;
    case 8:
        if (request("lasdpc10",8,Customer,0) == 0)
            schedule(9,Exponential(Ts4),Customer);
        break;
    case 9:
        release("lasdpc10", Customer);
        schedule(2, 0.0, Customer);
        break;
    case 10:
        if (request("lasdpc11",10,Customer,0) == 0)
            schedule(11,Exponential(Ts5),Customer);
        break;
    case 11:
        release("lasdpc11", Customer);
        schedule(2, 0.0, Customer);
        break;
    }

}

Close_Debug_File();
Close_Data_File();
}
/* ----- */

```

Figura C.6. Código do modelo do sistema distribuído com repetição