Pós-Graduação em Ciência da Computação

Avaliação de Desempenho do Serviço de Controle de Concorrência usando Redes de Petri Estocástica

por

Roberta Andrade de A. Fagundes

DISSERTAÇÃO DE MESTRADO

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, FEVEREIRO/2006

Universidade Federal de Pernambuco
Centro de Informática
Pós-Graduação em Ciência da Computação

ROBERTA ANDRADE DE ARAÚJO FAGUNDES

Avaliação de Desempenho do Serviço de Controle de Concorrência usando Redes de Petri Estocástica

Este trabalho foi apresentado à pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de mestre em Ciência da Computação.

ORIENTADOR: PAULO ROMERO M. MACIEL

CO-ORIENTADOR: NELSON SOUTO ROSA

RECIFE, FEVEREIRO/2006

Fagundes, Roberta Andrade de Araújo
Avaliação de desempenho do serviço de controle de concorrência usando Redes de Petri Estocástica. / Roberta Andrade de Araújo Fagundes. – Recife : O autor, 2006.
126 folhas: il., fig., tab.

Dissertação (mestrado) – Universidade Federal de Pernambuco. CIN. Ciência da Computação, 2006.

Inclui bibliografia.

1. Métodos Formais. 2. Avaliação de desempenho. 3. Serviço de controle de concorrência.

005.1 CDD (22.ed.) CIN2006-015

D

DEDICATÓRIA

Marluce e Fagundes, meus pais,

Rayssa, minha filha,

Paulo, meu esposo,

Leonardo, Rodrigo, João Paulo, meus irmãos,

Prof. Dr. Paulo Maciel, orientador e

Prof. Dr. Nelson Souto Rosa, co-orientador.

AGRADECIMENTOS

Obrigado meu Deus por mais esta conquista!

Gostaria de registrar meus sinceros agradecimentos como reconhecimento da dedicação de muitas pessoas que, direta ou indiretamente, contribuíram para a realização deste trabalho.

Primeiramente, ao meu orientador professor Paulo R. M. Maciel e ou meu co-orientador Nelson Souto Rosa, pelo enorme trabalho e tempo dedicado ao desenvolvimento desta pesquisa de forma objetiva e consistente.

A minha Família, meus pais, Marcule e Fagundes, pelo incentivo, ajuda e apoio moral. A meu esposo Paulo Mello, minha filha Rayssa Fagundes, pela paciência e compreensão.

Aos professores Djamel F. H. Sadok, e Gledson, pela participação da banca examinadora, contribuindo substancialmente para a melhoria da qualidade deste trabalho.

Aos colegas e professores do ALUPA (Alunos do Paulo), em especial a Fernando Carvalho, Raimundo Barreto, Sérgio Galdino, Socorro Vânia e todos os outros companheiros de mestrado que tanto deram apoio nos momentos em que parecia não haver luz no fim do túnel.

Aos demais professores e funcionários da UFPE.

SUMÁRIO

1. INTRODUÇÃO 14

1.1. Motivação 14

1.2. Proposta 16

1.3. Estrutura da dissertação 18

2. AVALIAÇÃO DE DESEMPENHO DE MIDDLEWARE BASEADA EM MODELOS 20

2.1. Serviço de eventos do CORBA 20

2.2. *Middleware* orientado a mensagem 22

2.3. Cliente/Servidor com redes token ring e csma/cd 24

2.4. Cliente/Servidor 26

2.5. CORBA 28

2.6. Considerações Finais 29

3. FUNDAMENTAÇÃO TEÓRICA 30

3.1. Avaliação de desempenho 30

3.2. Processo Estocástico 32

3.2.1. Cadeias de Markov 34

3.2.2. Métodos de soluções para probabilidade em estado estacionário 36

3.2.3. Métodos de solução para probabilidades em estado transiente 37

3.3. Redes de Petri 37

3.3.1. Componentes de uma rede de Petri 38

3.3.2. Redes Place/Transition 40

3.3.3. Rede de Petri marcada 41

3.3.4. Matriz de incidência 42

3.4. Redes de Petri e suas classes 44

3.5. Propriedades das Redes de Petri 45

3.5.1. Propriedades Comportamentais 45

3.5.2. Propriedades Estruturais 49

3.6. Método de Análise e Validação de RdP 50

3.7. Redes de Petri Estocásticas Generalizadas (GSPN) 51

3.8. Redes de Petri Estocástica Determinística (DSPN) 55
3.9. Distribuição por Fase 56
3.10. Simulação Estocástica de Eventos Discretos 60
3.11. Considerações Finais 61
4. MIDDLEWARE 63
4.1. Conceitos basicos 63
4.2. Middlware orientado a objetos 66
4.3. CORBA 67
4.4. Serviço de controle de concorrência 70
4.5. Considerações finais 73
5. METODOLOGIA DE AVALIAÇÃO 74
5.1. Visão geral da metodologia 74
5.2. Medição 77
5.3. Geração do modelo abstrato 80
5.4. Validação e análise quantitativas 80
5.5. Geração do modelo refinado 81
5.6. Validação quantitativa 81
5.7. Avaliação e análise dos resultados 82
5.8. Considerações finais 83
6. MODELOS PROPOSTOS 84
6.1. Definição do sistema e seus componentes 84
6.2. Modelo abstrato 85
6.3. Medição 87
6.3.1. Planejamento 88
6.3.2. Coleta dos dados 89
6.3.3. Análise dos dados 90
6.3.4. Recomendações 92
6.4. Definição das métricas 93
6.5. Modelos refinados 93
6.6. Validação 98
6.7. Considerações finais 101

7. ANÁLISE DOS EXPERIMENTOS 103
7.1. Introdução 103
7.2. Primeiro experimento 104
7.3. Segundo experimento 109
7.4. Terceiro experimento 112
7.5. Quarto experimento 114
7.6. Quinto experimento 116
7.7. Sexto experimento 118
7.8. Considerações finais 120
8. CONCLUSÕES E TRABALHOS FUTUROS 122
8.1. Trabalhos futuros 124
REFERÊNCIAS BIBLIOGRÁFICAS 125

LISTA DE FIGURAS

Figura 1-1 - Metodologia proposta para avaliação de desempenho. 17
Figura 2-1 - Arquitetura do Serviço de Evento do CORBA. 21
Figura 2-2 - Modelo Básico do Serviço de Evento do CORBA..... 21
Figura 2-3 - *Middleware* Orientado a Mensagem em Detalhe. 23
Figura 2-4 - Modelo GSPN básico do MOM. 23
Figura 2-5- Sistema Cliente Servidor para rede Token Ring. 26
Figura 2-6 - O modelo da conexão da mensagem para Windows baseado no cliente. 27
Figura 3-1 - Amostras de um processo estocástico. 33
Figura 3-2 - Elementos básicos de uma rede de Petri..... 39
Figura 3-3 - Períodos do dia representado com redes de Petri. 40
Figura 3-4 – Abstração x Interpretação 44
Figura 3-5 – Rede de Petri Marcada. 46
Figura 3-6 – Exemplo de rede não limitada. 47
Figura 3-7 – Rede segura depende da marcação inicial. 48
Figura 3-8 – Vivacidade. 48
Figura 3-9 - Geração do gráfico de alcançabilidade GSPN..... 54
Figura 3-10 - Diferentes tipos de distribuições. 57
Figura 4-1 - Visão geral de *middleware* 64
Figura 4-2 - Visão geral da comunicação em um *middleware* orientado a objetos..... 66
Figura 4-3 - Uma requisição sendo enviada através do ORB..... 68
Figura 4-4 - Estrutura das Interfaces do ORB. 69
Figura 4-5 - Definição da interface *Locks*et 72
Figura 5-1 – Diagramas de atividades. 75
Figura 5-2 - Fluxo da Medição. 77
Figura 5-3 - Fluxo da definição do desvio padrão. 79
Figura 6-1 - Identificação dos Componentes..... 84
Figura 6-2 - Modelo Abstrato. 87
Figura 6-3 - Processos Rodado no momento da Coleta de Dados..... 90

Figura 6-4 - Sub-rede Erlang. 93
Figura 6-5 - Modelo com Transição Determinística. 95
Figura 6-6 - Modelo com Transição Exponencial. 96
Figura 6-7 - Modelo Erlang. 97
Figura 6-8 - Modelo Simplificado no Timenet. 99
Figura 6-9a Tempo médio entre locks (Timenet) Figura 6.9b - Tempo médio entre locks(CORBA) 100
Figura 6-10 - Tempo Médio entre Locks usando Erlang. 101
Figura 7-1- Modelo Abstrato 104
Figura 7-2- Vazão do Sistema com a Técnica de Análise. 106
Figura 7-3- Vazão do Sistema com Erlang. 107
Figura 7-4- Vazão do *Middleware* com a Técnica de Análise. 108
Figura 7-5- Vazão do *Middleware* com Erlang 108
Figura 7-6- Vazão do Sistema com a Técnica de Simulação versus Clientes. 110
Figura 7-7 - Vazão do Middleware com a Técnica de Simulação versus Clientes. 111
Figura 7-8 - Vazão do *Middleware* versus Probabilidade de Leitura 112
Figura 7-9- Vazão do Sistema versus Probabilidade de Leitura. 113
Figura 7-10- Capacidade do servidor versus Taxa de Solicita;áo de Leitura. 114
Figura 7-11- Vazão do *Middleware* versus Taxa de Solicitação de Leitura. 115
Figura 7-12 - Vazão do Sistema versus Taxa de Solicitação de Leitura. 116
Figura 7-13 - Vazão do *Middleware* versus Probabilidade de Liberação do *lock*. 117
Figura 7-14 - Vazão do Sistema versus Taxa de Liberação do *Lock*. 118
Figura 7-15 – Vazão do Sistema versus Taxa do Servidor(Servindo) 119
Figura 7-16 - Vazão do *Middleware* versus Taxa do Servidor(Servindo). 119
Figura 7-17 - Capacidade do Servidor versus Taxa Servidor (Servindo) 120

LISTA DE TABELAS

Tabela 2.1 - Transições com suas taxas e função de guarda 22
Tabela 3.1 - Algumas interpretações típicas para lugares e transações. 39
Tabela 4.1 – Compatibilidade de *locks* 71
Tabela 5.1 – Protocolo de Medição. 78
Tabela 6.1 - Operação *lock* no modo de leitura 90
Tabela 6.2 - Operação *lock* sem *outliers* no modo de leitura. 91
Tabela 6.3 - Operação *unlock* no modo de leitura. 91
Tabela 6.4 - Operação unlock sem outliers no modo de leitura. 92
Tabela 6.5 - Fases do Novo_Lock e Unlock. 94

RESUMO

O processo de avaliação de desempenho pode ser implementado através de diversos métodos: medição que é processo de coleta de informações de um sistema real; simulação computacional e modelos analíticos que capturam o comportamento temporal de um sistema através de uma representação matemática. A avaliação baseada em modelos possibilita a análise de desempenho de sistemas, antes mesmo de sua implementação, o que possibilita ajustes ainda na fase de desenvolvimento. O uso de modelos como mecanismo de avaliação também torna possível a avaliação de cenários complexos, possibilitando, portanto, a análise de desempenho em função de restrições temporais e de recursos.

O interesse na avaliação de desempenho do sistema do *middleware* está aumentando. O CORBA é um padrão de *middleware* orientado a objetos definido pela OMG que permite aplicações distribuídas em uma rede (local ou mesmo na Internet) se comuniquem. O serviço de controle de concorrência (SCC) faz parte do conjunto de serviços conhecidos como serviços comuns do CORBA e são usados por várias aplicações de diversos domínios. O SCC do CORBA foi definido para coordenar o acesso a recursos compartilhados, através do uso de *locks*, garantindo a consistência quando o recurso é acessado concorrentemente.

Este trabalho propõe modelos redes de Petri estocástica para avaliar o desempenho do SCC do CORBA. Com a finalidade de validar os modelos propostos, os resultados da avaliação das redes de Petri são comparados com as medidas obtidas no OpenORB (núcleo do CORBA). Também são apresentados cenários de desempenho que auxiliam a tomada de decisões para melhoria do desempenho do SCC do CORBA.

Palavras Chaves: Métodos Formais, Avaliação de Desempenho e Serviço de Controle de Concorrência.

ABSTRACT

The performance evaluation process can be implemented through several methods: measurement, which is the process of collecting information from a real system; computational simulation; and analytical models that capture the temporal behavior of a system through a mathematical representation. Model-based evaluation allows the analysis of the system performance, even before its implementation, making adjusts possible during the development phase. The use of models as evaluation mechanisms also allows the evaluation of complex scenarios, enabling therefore the performance analysis according to constraints of time and resources.

The interest in the performance evaluation of the middleware system is increasing. CORBA is an object-oriented middleware standard, defined by OMG, which allows the communication among distributed applications in a local network or even the Internet. Concurrency control service (CCS) belongs to the set of CORBA common services, which are used by several applications of different domains. CORBA CCS was defined to coordinate the access to shared resources, through the use of locks, ensuring the consistency when the resource is accessed concurrently.

The present work proposes stochastic Petri nets models to evaluate CORBA CCS. Aiming at validating the proposed models, the Petri nets evaluation results are compared to measures obtained from OpenORB (CORBA core). We also present performance scenarios, which help decision-making to enhance CORBA CCS performance.

Key Words: formal method, Performance Evaluation, Concurrency Control Service.

1. INTRODUÇÃO

Este capítulo contextualiza avaliação de desempenho de middleware, apresentando os objetivos e contribuições alcançadas nesta dissertação. Por último é mostrada *estruturação dos demais capítulos.*

1.1. MOTIVAÇÃO

O rápido progresso em diferentes campos tecnológicos, ao longo dos últimos anos, tem proporcionado um crescimento notável em diversas áreas de aplicação da computação.

Muitos formalismos para descrição de sistemas têm sido desenvolvidos, mas poucos permitem a avaliação de desempenho [1]. As redes de Petri (RdPs) [2], [3] são um desses formalismos que permitem essa integração. RdPs é um termo genérico associado a um conjunto de formalismos matemáticos adequados para a modelagem de sistemas concorrentes, assíncronos, paralelos e distribuídos. O uso de RdP não está restrito à informações estruturais, também permite avaliar o comportamento e a dinâmica dos sistemas.

Um modelo adequado pode possibilitar a estimativa de métricas. Adicionalmente, podem refletir várias características e cenários associados, os quais podem ser difíceis de representar em um sistema real e principalmente quando este sistema ainda não está completamente desenvolvido.

O desenvolvimento de aplicações distribuídas tem sido uma tarefa cada vez mais complexa. Essas aplicações são normalmente compostas por elementos independentes que se encontram distribuídos em ambientes heterogêneos, e requerem uma infra-estrutura de comunicação sofisticada. Além disso, essas aplicações têm demandado a implementação de transações distribuídas, tolerância a falhas, segurança, transparência na comunicação e concorrência, dentre outros [16].

A infra-estrutura de comunicação mencionada é genericamente chamada de *middleware* [14]. O *middleware* tem o papel básico de esconder detalhes de comunicação de baixo nível e de tratar a heterogeneidade de software e hardware dos ambientes onde as aplicações distribuídas são executadas. Além disso, o *middleware* provê serviços (ex.: segurança, transação, concorrência, eventos etc) que agregam valor à interação entre as partes da aplicação. Em termos práticos, o *middleware* é o elemento responsável por permitir que as partes da aplicação interajam sem que o desenvolvedor tenha que tratar diretamente detalhes de comunicação de baixo nível.

Assim, problemas relativos à comunicação entre aplicações distribuídas surgiram e com isso, diversas categorias de *middleware* também, são eles: *middleware* transacional, *middleware procedural*, *middleware* orientado a objetos (MOO) e *middleware* orientado a mensagem. Cada uma dessas categorias essencialmente resolve um conjunto de problemas distintos e comuns às aplicações distribuídas, tais como comunicação assíncrona, processamento de transações, concorrência na comunicação etc.

Entretanto, o aumento de complexidade das aplicações distribuídas e o surgimento de novas tecnologias de desenvolvimento de sistemas, tais como orientação a objetos, fazem com que características adicionais, facilidades de utilização e otimizações dentro de MOOs sejam necessidades constantes. Além disso, o estudo da concorrência tem sido ponto chave para o aprimoramento dessas tecnologias pois, à medida que novos requisitos vão sendo adicionados às aplicações, domínios específicos que antes não utilizavam sistemas de *middleware* passam a utilizá-los de forma extensiva. Naturalmente, surgem também novas necessidades a serem incorporadas aos MOOs já existentes.

Um exemplo dessa adesão de domínios de aplicação à utilização de MOO é a rápida evolução dos sistemas de transferência eletrônica e de transações financeiras que, passaram de soluções pontuais a aplicações que necessitam se conectar com outros dispositivos e ter acesso a Internet. Esses novos requisitos levam à necessidade de uso de uma infra-estrutura de comunicação de

concorrência, normalmente já existente, mas que, precisa ser avaliada para caracterizar qual o impacto do desempenho do sistema para o domínio da aplicação.

Será mostrada nos capítulos seguintes a concorrência como característica adicional e de otimização do MOO para resolver alguns problemas relativos à comunicação entre as aplicações.

1.2. PROPOSTA

Em termos práticos, medidas de desempenho de *middlewares* são atividades complexas, pois demandam instrumentação sofisticada, são suportados por um conjunto restrito de ferramentas, agravados pela diversidade de operações dos sistemas de *middleware* atuais. Muitos formalismos de modelagem têm sido utilizados para a avaliação de desempenho de sistemas como [9], [12],[13]. Tais formalismos devem fornecer uma descrição dos aspectos do desempenho, relacionados aos sistemas modelados. Alguns formalismos fornecem uma representação gráfica dos sistemas. As redes de Petri estocásticas [5], [6],[7], [8], são exemplos de tais modelos e podem também ser aplicadas para verificação e/ou análise quantitativa das propriedades dos sistemas, tais como *liveness* e *boundedness* [2]. Redes Estocásticas são adotadas extensivamente para avaliar o desempenho de sistemas.

Este trabalho concentra-se na avaliação de desempenho do serviço de controle de concorrência do CORBA (SCC) através da proposição de modelos de redes de Petri estocástica que possibilitam avaliar a vazão do sistema e a vazão do *middleware*, como também, a verificação da quantidade de clientes aguardando para serem servidos, os dimensionamentos do *buffer*, facilitando, com isso, a tomada de decisões em relação ao sistema avaliado.

Além disso, uma metodologia foi proposta e pode ser resumidamente visualizada na Figura 1.1. Esta metodologia é composta por várias fases que vão desde o entendimento do sistema até apresentação dos resultados avaliados. As fases serão descritas em detalhes no Capítulo 5.

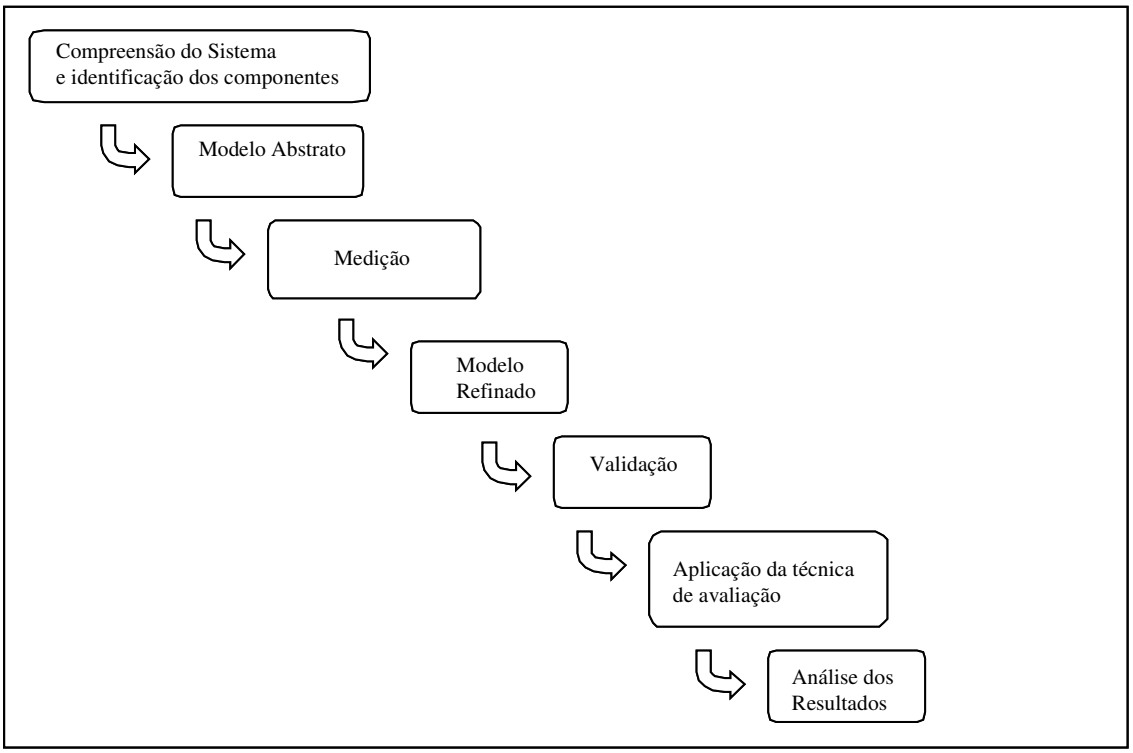


Figura 1-1 - Metodologia proposta para avaliação de desempenho.

A fase de compreensão do sistema e identificação dos componentes consiste em compreender o problema (sistema) a ser avaliado, sua relação com outros sistemas, os critérios respectivos que devem representar o desempenho do sistema e o destaque dos componentes que influenciam o desempenho do sistema. Na fase de geração do modelo abstrato, um modelo de desempenho inicial é gerado. Neste modelo, a informação temporal relacionada aos componentes do sistema não é expressa ainda, isto é, as transições temporais não possuem nenhuma distribuição de tempo específico - são transições temporais "genéricas". Na Fase de medição, o processo de medição dos componentes básico do sistema deve ser realizado de acordo com um protocolo especificado [39]. O conjunto de dados coletados, através da aplicação do protocolo, e um conjunto de estatísticas (média, desvio padrão, coeficiente de variação, etc.) é obtido. Assim, na fase de geração do modelo refinado, o processo de *moment matching* [15] é adotado para representar as distribuições genéricas de acordo com os valores relacionados aos coeficientes de variação [18]. As transições temporais genéricas são transformadas em sub-redes descrevendo

distribuições hiper-exponenciais, hipo-exponenciais e de Erlang; transições exponenciais ou determinísticas. Nesta fase, um conjunto de critérios de desempenho é especificado em termos de fórmulas probabilística que representam as vazões, as capacidades, os tempos médios, as disponibilidades etc. Os produtos desta fase são, conseqüentemente, modelos refinados. Na Fase de validação, um modelo refinado deve ser validado pela análise de um conjunto de cenários específicos em que as medidas reais são obtidas do sistema sob a avaliação e comparadas com os resultados respectivos fornecidos pela avaliação modelo. Na Fase de aplicação da técnica de avaliação, após terem sido gerados os modelos refinados, inicialmente, o método da avaliação deve ser escolhido (análise/simulação de estado estacionário ou transiente) de acordo com as métricas a serem avaliadas e a estrutura do modelo. A escolha destas métricas é feita de acordo com os critérios específicos que representam o desempenho do sistema. Na fase de análise dos resultados, após execução do método da avaliação, os resultados devem ser analisados com cuidado, interpretados, e ações podem ser recomendadas para ajuste do sistema.

1.3. ESTRUTURA DA DISSERTAÇÃO

Esta dissertação está organizada da seguinte forma: o Capítulo 2 apresenta os trabalhos relacionados com a avaliação de desempenho de *middleware* baseados em modelos. O Capítulo 3 define as redes de Petri, sua estrutura, propriedades e tipos. No Capítulo 4, apresentam-se os conceitos básicos de *middleware* e seus tipos. Uma maior ênfase é dada à categoria de *middleware* orientado a objeto, que é objetivo principal de estudo deste trabalho. Já o Capítulo 5 descreve a metodologia proposta para avaliação de desempenho, caracterizando suas fases. O Capítulo 6 propõe um modelo inicial que é validado, para que outros modelos sejam gerados, refletindo assim, o funcionamento real do serviço de controle de concorrência do CORBA. No Capítulo 7 são apresentados os resultados da avaliação de vários experimentos, através da descrição de vários cenários, onde os resultados são representados graficamente. No Capítulo 8 são apresentadas às conclusões obtidas durante o desenvolvimento desta

dissertação, como também, as principais contribuições do trabalho. Por fim, são apresentados trabalhos futuros que darão continuidade ao estudo desenvolvido.

2. AVALIAÇÃO DE DESEMPENHO DE MIDDLEWARE BASEADA EM MODELOS

O presente capítulo tem como objetivo apresentar os trabalhos relacionados à avaliação de desempenho de middleware. Estes trabalhos utilizam técnicas para a avaliação de desempenho de middleware, incluindo modelos analíticos, redes de Petri e Teoria das Filas.

2.1. SERVIÇO DE EVENTOS DO CORBA

Trivedi [24] avalia o desempenho de serviço de eventos do CORBA usando *Stochastic Reward Nets* (SRN) [25] para a modelagem. Os autores utilizam os modelos SRN para auxiliar a configuração de serviço de eventos. No serviço de evento do CORBA (Figura 2.1) o cliente e o servidor são conectados por canal. Os clientes conectam-se ao *proxy* servidor e os servidores conectam-se ao *proxy* cliente, ambos os objetos pertencem ao canal de eventos.

As especificações atuais de CORBA [11] não consideram os requisitos de confiabilidade ou falha. Atualmente alguns esforços têm sido aplicados no estudo e pesquisa em confiabilidade e disponibilidade em *middleware* CORBAs, principalmente por meio de replicação de objetos. A especificação do serviço de evento do CORBA (ver Figura 2.1) não requer garantias de entrega de nenhum evento. Assim, desenvolveram-se técnicas para recuperação de eventos perdidos, tais como re-envio do evento perdido e re-sincronização de informação entre os clientes e os servidores. Essa implementação típica não possui garantia que o evento foi entregue e é agravada com perdas de eventos, causando gargalos e tráficos durante a execução dos eventos.

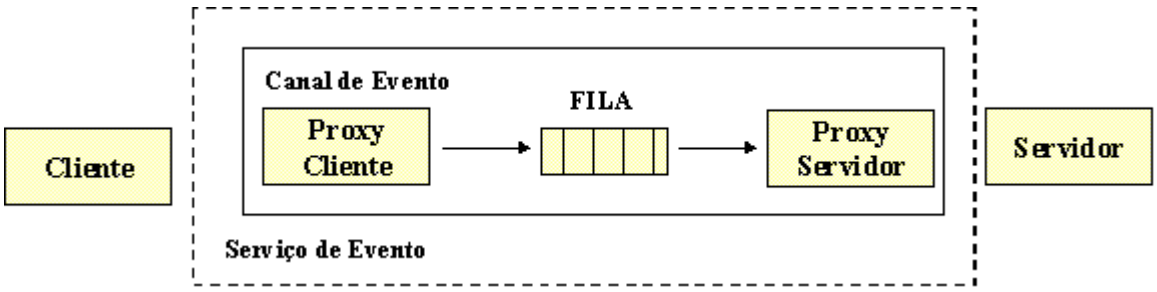


Figura 2-1 - Arquitetura do Serviço de Evento do CORBA.

A Figura 2.2 descreve um modelo SRN para o serviço de evento do COBRA. Os componentes (*Proxy* Cliente, Fila , *Proxy* do Servidor) principais são apresentados: O *Proxy* do Cliente é representado pelas transições tl e t2, e pelo lugar Pmmpp. Quando o lugar Pmmpp tem um *token*, a taxa da transição ts, (Ver Tabela 2.1) está ajustada como o ys1, e quando ts está vazio a taxa de disparo é ys2 . O arco do inibidor de Pmmpp a t2 impede o disparo de t2 quando Pmmpp tem um token, assegurando-se de que as transições tl e t2 não conflita. A Fila é representada pelo lugar PSQ, quando está cheio, os novos eventos que chegam são descartados, para solucionar esse problema um modelo resincronizado também foi proposto. O *Proxy* do Servidor é composto por dois lugares (Er_token_c e o Er_stage_c) que representam a espera de um evento para ser consumido.

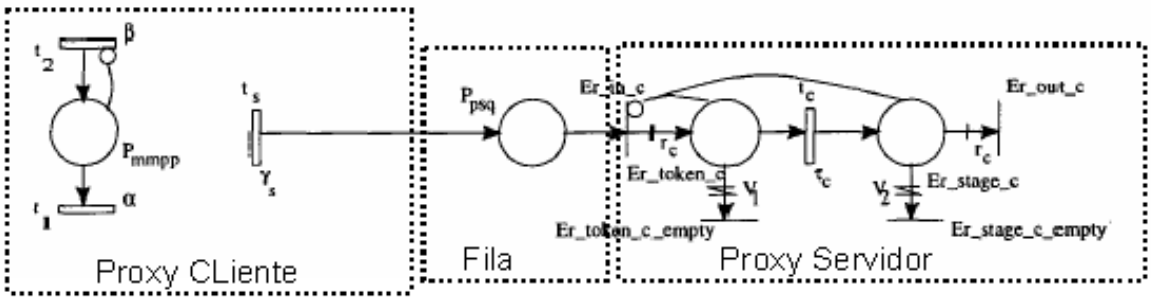


Figura 2-2 - Modelo Básico do Serviço de Evento do CORBA.

O peso de entrada do lugar Er_token_c e de saída do lugar Er_stage_c é representado por rc o número de fases da distribuição de Erlang e o tempo das transições exponenciais está apresentada na Tabela 2.1.

Tabela 2.1 - Transições com suas taxas e função de guarda

Transição	Taxa da Função	Função de Guarda
Ts	(#Pmmpp==1)? ys1 : ys2	-
Er_token_c_empty	-	#Ppsq + (Er_token_c + Er_stage_c)/rc > K?1:0
Er_stage_c_empty	-	#Ppsq + (Er_token_c + Er_stage_c)/rc > K?1:0

Algumas medidas de interesse foram calculadas para avaliar o desempenho do serviço de evento do CORBA de maneira a evitar as perdas e ocorrência de falhas. Essas medidas foram: a taxa de retenção, taxa de perda e a probabilidade de perda da mensagem, como também a probabilidade da fila está cheia.

Os modelos apresentados garantem a perda mínima do evento. Assim, os eventos perdidos são detectados e resincronizados, garantindo qualidade do serviço utilizado.

2.2. *MIDDLEWARE* ORIENTADO A MENSAGEM

Fernandes [16] propõe uma aproximação para modelar e analisar o desempenho do *middleware* orientado a mensagem (MOM) através das redes de petri estocásticas generalizadas (GSPN). Ao invés de adotar a técnica tradicional da medida para a análise de desempenho, concentra-se na modelagem do sistema do *middleware* usando GSPN seguido pela realização de experimentos. Os resultados obtidos da avaliação da rede de petri são comparados com as medidas do MOM. Adicionalmente, alguns resultados são apresentados a fim demonstrar os benefícios do modelo proposto. Esta pesquisa focaliza também em melhorar o desempenho do MOM sugerindo ajustes à arquitetura básica do MOM. Finalmente, indicam-se algumas decisões geralmente realizadas pelos administradores de sistema que podem ter um impacto principal no desempenho do MOM.

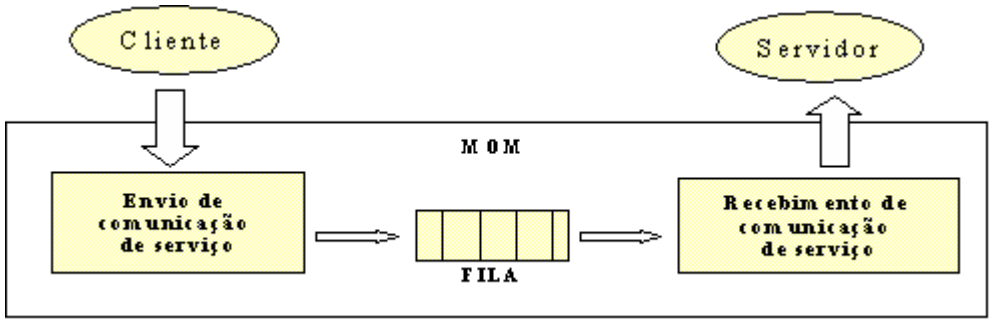


Figura 2-3 - *Middleware* Orientado a Mensagem em Detalhe.

Os MOMs fornecem o abstração de uma fila. Essa fila é usada para armazenar mensagens antes que sejam enviadas do cliente ao servidor (veja Figura 2.3). Um cliente coloca as mensagens na fila e o servidor conecta-se com seu servidor de comunicação que armazena suas mensagens em um tempo apropriado. Interagindo dessa maneira (comunicação assíncrona), clientes não esperam um reconhecimento do servidor e nem está ciente que o servidor está conectado ao MOM. .

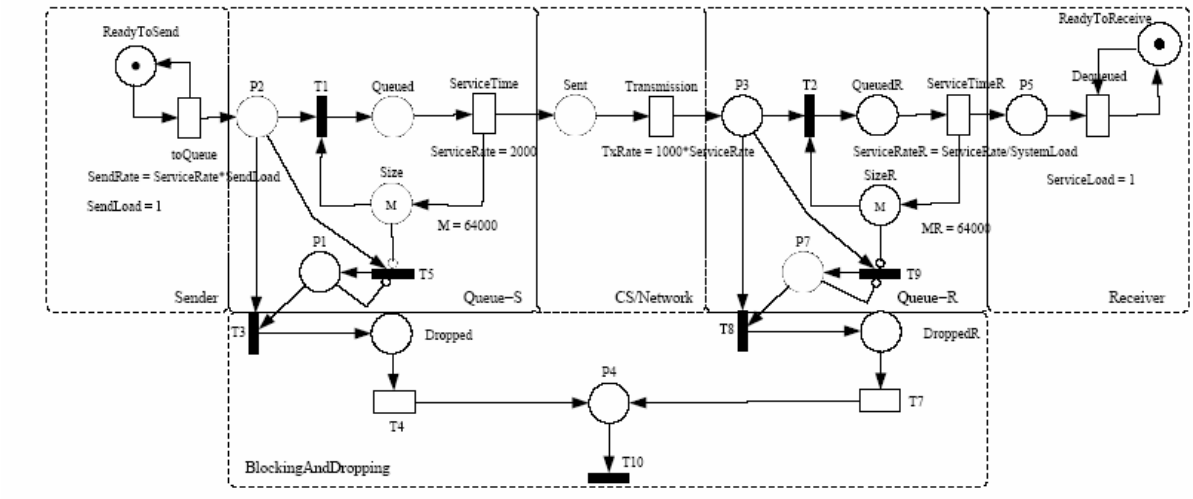


Figura 2-4 - Modelo GSPN básico do MOM.

A Figura 2.4 descreve um modelo GSPN do MOM. Os componentes principais (sub-redes: os subconjuntos dos lugares, das transições, suas relações e marcações) são nomeados: Sender (Cliente): esta sub-rede representa a carga gerada pela aplicação de envio; Queue-S(Fila do Cliente): este modelo descreve o principal buffer do MOM no lado do cliente; CS/Network (Rede): esta sub-rede

modela o canal de comunicação e o serviço de comunicação; Queue-R (Fila do Servidor): este modelo representa o principal buffer do MOM no lado do servidor; Receiver (Servidor): esta sub-rede descreve a taxa de serviço da aplicação servidora; e *BlockingAndDropping*: esta sub-rede representa as mensagens perdidas.

As contribuições deste trabalho incluem: geração de modelos de redes de Petri estocásticas MOM, avaliação de desempenho do MOM sob os vários parâmetros de carga de trabalho que conduzem às introspecções a respeito da arquitetura do MOM. Os benefícios incluem também a possibilidade de planejamento de capacidade do *middleware*, ajudando ao estudo das especificações de MOM (por exemplo, JMS). As redes de petri estocásticas foram adotadas, pois são difundidas e apresentam um formalismo apropriado para a avaliação de desempenho do sistema, permitindo a modelagem natural de vários elementos do MOM (por exemplo, filas, clientes e aplicação servidora) e diversos aspectos dinâmicos (por exemplo, operação assíncrona, tamanhos da mensagem, falhas, qualidade de níveis do serviço (QoS), capacidade etc.)

2.3. CLIENTE/SERVIDOR COM REDES TOKEN RING E CSMA/CD

Trivedi et al [17], propõe modelos para avaliação de desempenho de sistema Cliente/Servidor com redes *CSMA/CD* e *Token Ring*. Para realizar essa avaliação a técnica utilizada foi às redes de Petri estocásticas (SPN).

A rede *CSMA/CD* utilizada por uma estação de trabalho para transmissão de mensagem, é composto por um conjunto de passos. Primeiro escuta o meio, se estiver livre, transmite a mensagem. Segundo, caso esteja ocupado, a estação continua escutando até desocupar, para que a mensagem possa ser transmitida. Terceiro, se mensagem transmitida é envolvida em colisão, a estação espera um tempo aleatório e volta a escutar o meio novamente para verificar quando o meio está livre.

A rede *Token Ring*, utilizada por uma estação de trabalho para transmissão de mensagem, possui um conjunto de passos a ser seguido. Primeiro a estação

pega o *token* para transmitir uma mensagem. Segundo, caso não consiga, espera até que o *token* seja liberado, para transmitir uma mensagem. Terceiro, se mensagem transmitida é envolvida em colisão, a estação espera um tempo aleatório e volta a pegar um *token* para poder transmitir uma mensagem.

A Figura 2.5 é descrita pelo seguinte: o lugar PT1 contém um *token* e representa o estado ocioso do cliente. A taxa de disparo da transição temporizada tta é λ. Assim, o disparo de tta representa o evento de solicitação gerado pelo cliente. O lugar PTA representa o estado do cliente esperando por um *token* na rede para fazer a solicitação. O lugar PTS representa que o *token* está livre na rede. Suponha que exista um *token* em PTA. Então, a transição temporizada tts está habilitada e representa o tempo médio de transmissão da solicitação, senão a transição imediata s1 dispara. O lugar PSq representa que o cliente finalizou a transmissão da solicitação, isso se dá, através do disparo de tts. Caso o cliente não tenha transmitido a solicitação, então s1 está habilitada. A transição temporal tsp representa que o *token* chegou na rede e é passado para seus vizinhos. O lugar PS1 representa que a solicitação do cliente chegou no servidor. O lugar PTW representa o estado do cliente esperando por uma resposta do servidor. O lugar PSS representa o servidor em estado pronto, assim quando receber um *token* da rede, já pode começar a transmitir a resposta, então, as estações que já finalizaram sua transmissão liberam o *token* na rede.

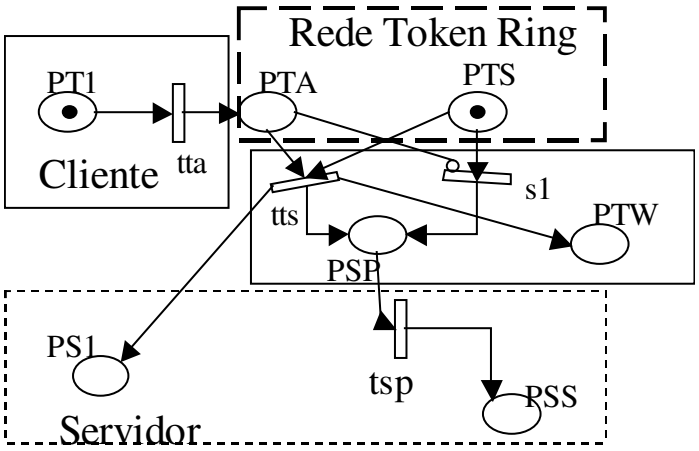


Figura 2-5- Sistema Cliente Servidor para rede Token Ring.

As medidas calculadas são: o tempo médio de resposta e a vazão são avaliados para um sistema cliente servidor baseado na rede *Token Ring* e *CSMA/CD*. Portanto os modelos gerados para redes *Token Ring* e *CSMA*/*CD* geram um número grande de espaço de estados, soluções computacionais são requeridas com 5, 7 e 10 estações de trabalho. Entretanto, modelos com um número maior de estações são facilmente construídos, devido à facilidade de extensão do modelo para um sistema com mais de um servidor. Outros resultados são para sistemas no qual o tempo médio para geração da solicitação/resposta e o tempo de transmissão da mensagem são distribuições exponenciais. Entretanto, podem-se modelar sistemas com distribuição determinísticas, Erlang. Portanto, recomenda-se a utilização de aproximação por fase, reduzindo o espaço de estado sem ocorrência significativa de erro.

2.4. CLIENTE/SERVIDOR

Jongwook Kim e Young Hee Lim [19] propuseram um simples modelo de redes de Petri temporizadas baseado nos resultados dos experimentos. A simples análise pode ser feita com este modelo, a fim resolver o problema de alocação de recurso. O modelo representado na Figura 2.6 é composto por 8 lugares e 5 transições. Onde: P(i,0): representa o estado do cliente i quando solicita transferência de mensagem ao servidor. P(i,1): representa o estado do cliente i

quando o tempo conexão acabou. P(i,2): representa o estado do cliente quando perdeu a conexão com o servidor. P(i,3) : representa o estado do cliente i quando espera por uma disponibilidade na rede. P(i,4): representa o estado do cliente i quando é notificado com tempo acabado. P(i,5): representa o estado do cliente i quando emite mensagens sucessivas ao servidor. P(i,6): representa o estado do cliente i quando recebe sucessivas mensagens do servidor. t(i,0): representa a ação do cliente i quando inicia o envio de solicitações de mensagens ao servidor; t(i,1) representa a ação do cliente i quando o tempo de conexão com rede acabou; t(i,2) representa a ação da rede em enviar uma mensagem de tempo esgotado para o cliente; t(i,3): representa a ação do cliente i obter conexão na rede t(i,4): representa a ação do cliente i iniciar o recebimento de dados na rede;Pn,: representa o estado quando a rede está disponível.

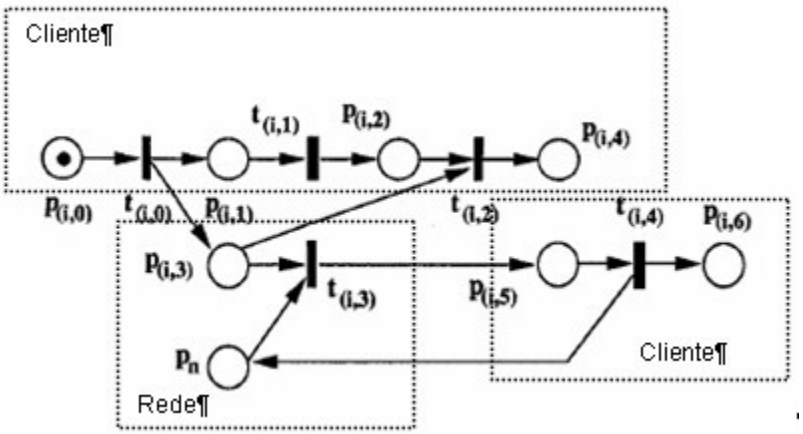


Figura 2-6 - O modelo da conexão da mensagem para Windows baseado no cliente.

Os autores conduziram vários experimentos. Para os experimentos, não havia nenhuma atividade no tempo de processador do servidor, mas encontrou-se que a rede é o gargalo do qual pode resultar na falha de trabalho inteiro. O modelo sugerido não é um mapeamento direto do protocolo de camada baixa e do comportamento direto do servidor, mas reflete fielmente o aspecto funcional do sistema Windows baseado no cliente.

2.5. CORBA

Dorina Petriu, Hoda Amer, Shikharesh Majumdar, Istabrak Abdull-Fatah [22] propõem a modelagem analítica de sistemas baseados em *middleware* que podem ser usados no desempenho de aplicações cliente-servidor baseado em CORBA. Aplicam-se modelos de redes de filas (LQN) para dois tipos de arquiteturas de *middleware*: *handle-driven e forwarding*. As saídas dos modelos são comparadas com os valores medidos para determinar a exatidão da técnica de modelagem.

A construção de uma aplicação cliente-servidor, um cliente executa um ciclo repetidamente. Em cada ciclo faz uma solicitação ao servidor A e ao servidor B. Executando uma operação de bind antes de cada pedido. A comunicação síncrona é usada: depois que uma solicitação é feita, o cliente permanece bloqueado até que a resposta seja recebida. Quando um serviço é solicitado por servidor particular, o servidor executa um laço e consome uma quantidade pré-determinada de tempo da unidade central de processamento (CPU). A aplicação sintética é usada porque fornece a flexibilidade de experimentos com vários níveis de parâmetros diferentes de carga de trabalho, tais como, o tempo do serviço em cada servidor e o retardo nó intermediário (*inter-node)*. Tal aplicação é apropriada para investigar a exatidão dos modelos analíticos proposto. A aplicação sintética é caracterizada por um número de parâmetros que são: *Service Demands* (SA, SB): o tempo requerido pelo servidor A ou pelo servidor B respectivamente para fornecer o serviço solicitado. Sempre que um servidor em particular A (ou B) é invocado para consumir SA (ou o SB) unidades de tempo da CPU. *Inter-Node Delay* (D): os clientes, o agente, e os servidores podem ser separados por um número nó intermediário. O atraso do nó intermediário entre cliente e servidor, entre servidor e agente, assim como, entre cliente e agente são caracterizados por D; *Message Length* (L): o tamanho da mensagem, L, enviado pelo cliente para fornecer um nome do método junto com sua lista de argumento ou enviado pelo servidor para retornar os resultados.

As medidas de desempenho calculadas foram: tempos de resposta e vazão, específicas para diferentes combinações do SA, SB, L e D. Essas medidas

foram repetidas para produzir os intervalos de confiança de 95%. Os resultados encontrados que modelo de LQN são razoavelmente aceitáveis, pois os erros são menos de 12%. O gargalo no sistema foi predito também corretamente. O maior erro de -18% ocorreu quando ambas a demanda do serviço nos servidores e a população eram grandes. Nesses casos, o modelo analítico tende a predizer uns tempos de resposta mais curtos porque não pode capturar do "um efeito combóio" que apareça no sistema real devido a seu forte comportamento determinístico (os processos seguem o mesmo teste padrão de solicitação e as demandas do serviço são determinísticas). O modelo de LQN não pode capturar este efeito devido as suas suposições dos tempos de serviço são distribuídos exponencialmente e a ordem das visitas é aleatória. Apesar daquelas diferenças, o modelo de LQN pode ser completamente útil no desempenho de aplicações baseada em CORBA, devido à velocidade e facilidade do uso.

2.6. CONSIDERAÇÕES FINAIS

Este capítulo apresentou avaliação de desempenho de *middleware*. Deste estudo, identificam-se dois principais propósitos do CORBA: o provimento de um padrão de uso dos MOOs, facilitando a incorporação dos mesmos a sistemas e contextos e permitindo até mesmo uma facilidade de comunicação sem maiores problemas, já que o CORBA nada mais é do que um *middleware* orientado a objetos que possui uma interface de comunicação tanto com cliente quanto com servidor. Portanto, os trabalhos apresentados servirão como embasamento para realização do estudo desenvolvido neste trabalho (avaliação de desempenho do serviço de controle de concorrência do CORBA utilizando redes de Petri estocástica).

3. FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os principais conceitos sobre redes de Petri estocástica generalizada (GSPN), assim como, características, propriedades e sua aplicabilidade. Primeiro, é apresentado os conceitos básicos sobre avaliação de desempenho e processos estocásticos. Posteriormente, são introduzidas as cadeias de Markov de tempo contínuo onde são apresentados os métodos de análise transiente e estacionária. Faz-se também uma revisão bibliográfica sobre redes de Petri.

3.1. AVALIAÇÃO DE DESEMPENHO

Para efetuar uma avaliação de desempenho [36], [38], além de escolher e reconhecer a técnica a ser aplicada, outros aspectos também devem ser considerados. É importante definir e seguir um esquema sistemático, que descreva todos os passos envolvidos em cada etapa do processo de avaliação. Pode-se dizer então que, avaliação de desempenho de sistemas computacionais, consiste em um conjunto de técnicas e metodologias que permitem responder a questão de como se obter um alto desempenho de um sistema computacional a um baixo custo. A atividade de avaliação de desempenho compreende, entre outras:

- Seleção de técnicas de avaliação, métricas de desempenho e cargas de trabalho;
- A correta condução das medidas de desempenho e das simulações;
- A utilização de técnicas de avaliação apropriadas para comparar as diversas alternativas;
- O desenvolvimento dos experimentos de medição e simulação visando prover a maior quantidade de informação com o menor tempo;
- A utilização de modelos para analisar o desempenho dos sistemas.

Contrário ao pensamento comum, a avaliação de desempenho não pode ser realizada de forma automatizada. Cada avaliação exige um conhecimento dos sistemas a serem modelados e uma cuidadosa seleção da metodologia, carga de trabalho e ferramentas. Definir o problema real e convertê-lo na forma onde as ferramentas e as técnicas usuais podem ser utilizadas.

As técnicas de avaliação de desempenho podem ser utilizadas em duas situações distintas. Dentre as técnicas comumente utilizadas para este propósito, podemos destacar: medições, simulações e modelagem analítica.

A medição pode ser aplicada quando o programa estiver totalmente implementado. Normalmente, utiliza-se medição em conjunto com outras técnicas de modelagem, avaliação e predição de desempenho. Deve-se considerar, ao planejar uma medição de desempenho, o propósito da medição, a seleção da carga de trabalho e sua implementação, os dados a serem coletados, a instrumentação utilizada na coleta destes dados e a forma de validação dos resultados. Todos os experimentos de medição de desempenho requerem três elementos básicos: o sistema sob teste, os geradores de carga de trabalho e a instrumentação que irá coletar os dados de desempenho.

A simulação é utilizada tanto em avaliação de desempenho, quanto na validação de modelos analíticos. Ao contrário das medições, as simulações baseiam-se em modelos abstratos do sistema, logo, não exige que o sistema esteja totalmente implementado para que seja aplicada. Assim, os modelos utilizados durante a simulação são elaborados através da abstração de características essenciais do sistema, sendo que a complexidade e o grau de abstração do mesmo pode variar de um sistema para outro. Durante a simulação controla-se com maior eficiência os valores assumidos por parâmetros do sistema. Com isso, fica mais fácil obter informações relevantes para a avaliação de desempenho. Através de simulações, pode-se verificar e prever o comportamento de programas sobre arquiteturas, ainda não utilizadas em testes.

A modelagem analítica utiliza um conjunto de equações e funções matemáticas para descrever o comportamento de uma aplicação. Os fatores que influenciam e interferem no comportamento da aplicação são modelados e

representados através dos parâmetros de equações matemáticas. Essas equações são chamadas de modelos analíticos [37]. Apesar desses modelos considerarem parâmetros específicos de uma arquitetura, podem ser facilmente adaptados para outras plataformas. Durante a construção dos modelos analíticos, deve-se levar em consideração a sua complexidade e praticidade. Os modelos analíticos permitem uma análise ampla e aprofundada em relação aos efeitos causados pelos parâmetros definidos nas equações sobre a aplicação. Além disso, também se pode estabelecer possíveis relacionamentos entre cada um dos parâmetros considerados. Essa modelagem, quando comparada às demais técnicas de avaliação de desempenho, apresenta menor custo de execução. Para validar os resultados alcançados através dos modelos elaborados, a modelagem analítica pode compará-los aos valores reais medidos em testes experimentais. Esses valores poderão comprovar as predições realizadas através dos modelos analíticos.

3.2. PROCESSO ESTOCÁSTICO

A avaliação de desempenho e confiabilidade de sistemas, através de métodos analíticos, tem sido objeto de numerosos estudos nas áreas de ciência da computação e matemática aplicada. Processos estocásticos são modelos matemáticos úteis para a descrição de um fenômeno de natureza probabilística, como uma função de um parâmetro que, usualmente, tem o significado de tempo.

Um processo estocástico {X(t), t Є T} é uma família de variáveis aleatórias definidas sobre o mesmo espaço de probabilidades, indexadas por um parâmetro t e assumindo valores no espaço de estado S. Geralmente, t significa tempo, T descreve intervalo de tempo, e o espaço de estado S compreende o conjunto dos possíveis valores de X(t) [23].

Classificam-se os processos estocásticos de acordo com o tipo da variável aleatória associada ao processo. Os processos estocásticos de tempo contínuo são descritos por variáveis aleatórias de tempo contínuo e seus espaços de estados podem ser discretos ou contínuos. Os processos estocásticos de tempo

discreto são descritos por variáveis aleatórias de tempo discreto, e, da mesma maneira, seus espaços de estados podem ser discretos ou contínuos.

A Figura 3.1 mostra a representação gráfica de dois processos estocásticos: (a) de tempo contínuo e (b) de tempo discreto. Ambos têm espaços de estado discreto.

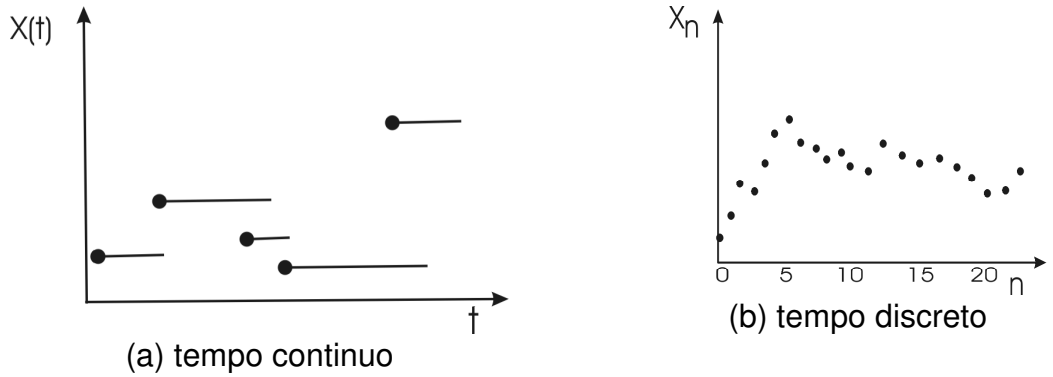


Figura 3-1 - Amostras de um processo estocástico.

Em geral, a descrição probabilística completa de um processo aleatório genérico não é factível. Processos Markovianos são uma classe especial de processo estocástico, para o qual a descrição probabilística é simples e de relevância particular.

Os Processos Markovianos são definidos como processos estocásticos que satisfazem às propriedades Markovianas:

M1 – A evolução futura dos estados não depende dos estados anteriores;

M2 – A evolução futura dos estados não depende do tempo de permanência no estado atual.

As propriedades Markovianas também são conhecidas por ausência de memória. Os processos Markovianos de tempo contínuo e espaço de estado discreto são denominados Cadeias de Markov de Tempo Contínuo (CTMC). O processos Markovianos com espaço de estado discreto e tempo discreto são denominados de Cadeias de Markov de Tempo Discreto (DTMC) [27].

3.2.1. Cadeias de Markov

O modelo markoviano herdou esse nome pelo trabalho pioneiro desenvolvido pelo matemático russo A.A. Markov no início do século XX. De fato, seu trabalho iniciou os trabalhos na área dos processos Estocásticos. Nas primeiras duas décadas do século XX, o matemático Danish A. K. Erlang aplicou técnicas Markovianas para resolver problemas de planejamento de capacidade para a Companhia de Telefone Copenhagen. Seu modelo foi adaptado por outros, entre eles, o Britisch Post Office. O matemático russo A. N. Kolmogorov consolidou as cadeias de Markov nos anos 30 do século passado.

Durante todo o século XX, o trabalho desses pioneiros se tornou mais compreendido e difundido. Atualmente, os processos estocásticos e as Cadeias de Markov têm importância fundamental para as avaliações de desempenho e confiabilidade de sistema em muitas áreas da ciência e engenharia. Por exemplo, na biologia têm sido utilizados para modelar o crescimento e a redução das populações; na física, para modelar iterações entre partículas elementares; na engenharia química, para modelar as reações (cadeias) entre moléculas ou para modelar processos de mistura; na ciência de gerência, para modelar o fluxo dos produtos ou sistemas de produção flexível ou para modelar a disponibilidade de linhas de produção e, principalmente, em computadores, na ciência e na engenharia de comunicação, para modelar o desempenho e a dependência do sistema em uma variedade de parâmetros.

As Cadeias de Markov de Tempo Contínuo são uma classe de processos estocásticos com espaço de estado enumerável, onde as variáveis aleatórias {X(t): t Є T} são de tempo contínuo e satisfazem às propriedades markovianas, isto é, ausência de memória.

A representação padrão de uma CTMC é feita usando-se diagramas de transições entre estados, isto é, com grafos dirigidos rotulados. Os estados da CTMC são representados pelos vértices. Os arcos são rotulados às taxas de transição correspondentes entre estados (l). Em uma CTMC, as probabilidades de transição de estado são calculadas pela equação:

$$p_{i,j}(s,t) = P\{X(t) = j \mid X(s) = i\}$$

O comportamento do processo markoviano é descrito pelas equações de probabilidade de transição de Champman-Kolmogoroff [10].

$$p_{i,j}(s,t) = \sum_{todo\ r} p_{i,r}(s,u) \cdot p_{r,j}(u,t), \qquad s \le u \le t$$

Onde i, r e j são estados e s, u, e t instantes de tempo.

Desenvolvendo-se essa equação (na notação matricial), obtém-se a equação diferencial:

$$\frac{d\pi(t)}{dt} = \pi(t)Q$$

Onde p(t) – é o vetor das probabilidades de estado i no tempo t;

p(0) – é o vetor das probabilidades de estado no tempo inicial;

Q – é a matriz das taxas de transição entre os estados;

$\frac{d\underline{\pi}(t)}{dt}$ - é a primeira derivada do vetor probabilidade com relação ao tempo.

A solução transiente, ou dependente do tempo, é importante quando o sistema a avaliar é dependente do tempo. Para modelos ergódicos [26], considerando tempos de execução longos, pode-se mostrar que a probabilidade dos estados converge para valores constantes [10]. O comportamento transiente da cadeia de Markov nos fornece uma informação precisa de desempenho sobre os instantes iniciais do sistema. Assumindo-se que a probabilidade p(t) é independente do tempo, isto é, $\pi_i = \lim_{t\to\infty} \pi_i(t)$ (homogeneidade), conseqüentemente, p'(t) = 0, portanto:

$$\underline{\pi}Q = \underline{0}, \qquad \sum_{i=1}^{N} \pi_i = 1$$

A equação da direita é a condição de normalização adicionada para assegurar que a solução obtida é um único vetor de probabilidade. A parte esquerda, sozinha, teria um conjunto de soluções infinitas. Normalizando as soluções, chega-se em um único vetor de probabilidades.

3.2.2. Métodos de soluções para probabilidade em estado estacionário

Para computar as probabilidades em estado estacionário de uma CTMC finita, com N estados, faz-se necessário resolver o sistema de N equações lineares.

$$\underline{\pi}Q = \underline{0}, \qquad \sum_{i=1}^{N} \pi_i = 1$$

Assumindo-se que a cadeia de Markov é irredutível e que p existe e é independente de p(0), para se solucionar o sistema de equações, dois tipos de métodos de solução podem ser escolhidos: Métodos Diretos e Métodos Iterativos [68]. Um método é chamado direto quando fornece a solução exata desejada, após um número finito de passos. Um método é chamado iterativo quando fornece uma seqüência de soluções aproximadas que convergem para o valor exato.

A principal característica dos chamados métodos diretos é que eles focam na reescrita do sistema de equações de tal forma que se obtêm expressões explícitas para o cálculo das probabilidades em estado estacionário. E podem ser: Decomposição LU [21] e a Eliminação Gaussiana[15] . Por razões computacionais e eficiência de memória, os métodos diretos não são utilizados quando o número de estados é maior que mil (1000). Nestas condições, utilizam-se os métodos iterativos.

Os Métodos Iterativos [15], [21] são métodos numéricos para resolução de um sistema linear, que geram uma seqüência de vetores $\{x^{(k)}\}$ a partir de uma aproximação inicial $x^{(0)}$. Sob certas condições, essa seqüência converge para uma solução, caso ela exista. Os procedimentos iterativos não resultam em uma solução explícita do sistema de equações; não sendo possível estimar quantas interações são necessárias para se obter a precisão desejada. Os métodos iterativos mais conhecidos são: Método da Potência, Método de Jacobi e Gauss-Seidel, Método SOR que podem ser encontrados em detalhes [15] e [21]. Muitas métricas podem ser avaliadas através de análise estacionária, contudo quando é necessário avaliar critérios dependentes do tempo e do estado inicial ou quando

uma solução estacionária não é alcançada. Probabilidades dependentes do tempo devem ser obtidas.

3.2.3. Métodos de solução para probabilidades em estado transiente

As probabilidades de estado transiente de um CTMC são especificadas por um sistema de equações diferenciais de primeira ordem:

(2) $$\underline{\pi}'(t) = \underline{\pi}(t)\, Q, \quad dado\ \underline{\pi}(0)$$

As probabilidades especificadas em termos de determinam as probabilidades de se estar em determinado estado em um determinado instante de tempo. Existem diversos métodos de resolução da equação diferencial (2) para computação das probabilidades transientes.

A solução numérica de sistemas de equações diferenciais tem sido, desde muito tempo, um tópico importante em matemática numérica. Muitos procedimentos numéricos têm sido desenvolvidos para essa finalidade. Entre eles, os mais utilizados são: Métodos de Runge-Kutta [28] e Uniformização [1],[19].

3.3. REDES DE PETRI

As redes de Petri (RdP) [29] devem o seu nome ao trabalho de Carl Adam Petri que, na sua dissertação de doutoramento[30], submetida, em 1962, à Faculdade de Matemática e Física da Universidade Técnica de Darmstadt, na Alemanha, apresentou um tipo de grafo bipartido1 com estados associados, com o objetivo de estudar a comunicação entre autômatos [3]. O seu desenvolvimento posterior foi catalisado devido ao potencial de modelagem, sincronização de processos, concorrência, conflitos e partilha de recursos. Desde então, trabalhos teóricos e aplicações têm sido desenvolvidos tanto com relação ao desenvolvimento de técnicas de análise quanto com relação ao desenvolvimento de variantes ao modelo seminal, tendo em vista aplicações específicas.

Como ferramenta matemática e gráfica, as RdP oferecem um ambiente uniforme para a modelagem, análise formal e simulação de eventos discretos,

permitindo uma visualização simultânea da sua estrutura e comportamento. Mais especificamente, as RdP modelam dois aspectos dos sistemas, eventos e condições, bem como as relações entre eles. Como veremos, é possível relacionar, de forma intuitiva, condições e eventos com os dois tipos de nós da rede, respectivamente lugares e transições. As vantagens da utilização das RdPs na modelagem de sistemas são conhecidas [2]:

- RdPs fornecem uma formalismo de modelagem que permite uma representação gráfica e é fundamentado matematicamente.
- Existe uma grande variedade de algoritmos para o projeto e análise de RdPs além de ferramentas de software desenvolvidas para auxiliar neste processo.
- RdPs provêem mecanismos para abstração e refinamento que são integrados ao modelo básico.
- Existe um grande número de ferramentas, tanto comerciais quanto acadêmicas, para o projeto, simulação e análise de sistemas baseados em RdPs.
- RdPs têm sido utilizadas em muitas áreas distintas e em conseqüência existe um grande número de especialistas no campo da modelagem.
- Existem várias extensões ao modelo básico de RdP.

3.3.1. Componentes de uma rede de Petri

O grafo de uma rede de Petri é direcionado e bipartido, possui pesos e dois tipos de vértices, chamados lugares (*places*) e transições (*transition*) com arcos que vão de um lugar para uma transição ou de uma transição para um lugar.

Graficamente, representam-se lugares com círculos e transições com barras ou retângulos (ver Figura 3.2). Os arcos são rotulados com seus pesos, exceto se o peso for rotulado com valor um (monovalorado), e um arco com peso k, deve ser rotulado como k arcos paralelos.

Figura 3-2 - Elementos básicos de uma rede de Petri.

Lugares e transições representam respectivamente condições e eventos. Os lugares, portanto, são ditos componentes passivos e as transições são componentes ativos. Uma transição possui um certo número de lugares de entrada e saída, o que representa as pré e pós-condições do evento observado. A realização de uma ação, portanto, está associada a algumas pré-condições, ou seja, existe uma relação entre os lugares e as transições, que possibilita a realização de uma ação. De forma semelhante, após a realização de uma ação, alguns lugares terão suas informações alteradas (pós-condições). Algumas interpretações típicas de transições e seus lugares de entrada e saída são mostradas na Tabela 3.1.

Tabela 3.1 - Algumas interpretações típicas para lugares e transações.

Lugares de entrada	Transição	Lugares de Saída
Pré-condições	Evento	Pós-condições
Dados de entrada	Passo computacional	Dados de saída
Sinais de entrada	Processador de sinais	Sinais de saída
Recursos requeridos	Tarefa	Recursos liberados
Condições	Cláusula lógica	Conclusões
Buffers	Processador	Buffers

Dado os conceitos informalmente apresentados sobre redes de Petri ilustramos a seguir, segundo [5], um pequeno exemplo para entendimento: o ciclo repetitivo dos turnos (períodos) de um dia. Dividamos o dia em três períodos: manhã, tarde e noite, ou seja, há três condições. A transição de uma dessas condições para uma outra, por exemplo - amanhecer (noite ➔ manhã), são os eventos. O modelo que representa o ciclo operacional desse sistema é formado pelas três condições, representadas por três variáveis de estado (lugares), e por

três eventos (transições): amanhecer, entardecer e anoitecer. Para representar a situação atual, ou seja, em que a condição encontra-se o sistema modelado, usa-se uma marca grafada (um ponto) no lugar que corresponde a essa situação, por exemplo: a condição atual é manhã. Na Figura 3.3a tem-se o modelo que representa este sistema e a sua situação atual.

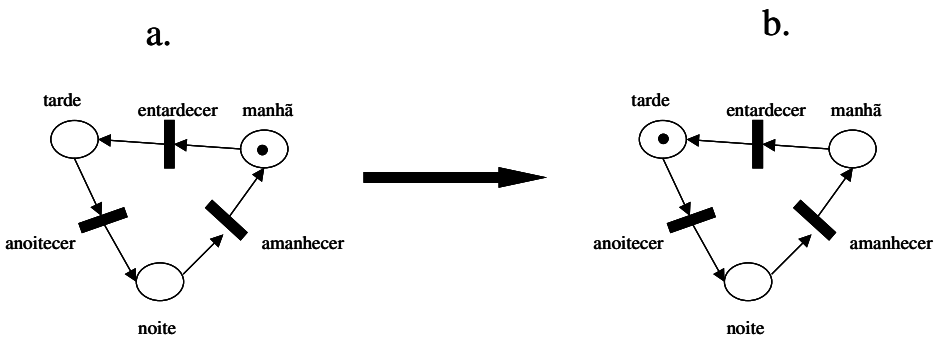


Figura 3-3 - Períodos do dia representado com redes de Petri.

Nesse modelo, temos a condição atual representada pela marca no lugar manhã. Estando nesta condição o único evento que poderá ocorrer é entardecer, que é representado pela transição de mesmo nome. Na ocorrência desse evento, teremos uma nova situação atual, ou seja, tarde, que é representada graficamente na Figura 3.3b por uma marca no lugar tarde.

3.3.2. Redes Place/Transition

Pode-se definir mais formalmente as redes de Petri de três maneiras distintas. Uma sobre o ponto de vista da teoria *bag* (é uma generalização do conceito de conjunto que admite a repetição de elementos) [2], que apresenta mapeamentos das transições para os *bags* de lugares. Um segundo tipo de definição usa a notação matricial. E a terceira, define as Redes de Petri através de relações e pesos associados a estas relações. Neste trabalho será apresentado apenas o segundo tipo de definição, a notação matricial.

Devido aos resultados existentes nos estudos da álgebra matricial, as redes de Petri utilizam este ferramental para a formalização de sua teoria, possibilitando

a análise de propriedades comportamentais e estruturais, posteriormente apresentadas neste capítulo.

Segundo o ponto de vista matricial, a estrutura das redes de Petri é apresentada por uma quíntupla formada pelo conjunto de lugares, o conjunto de transições, a matriz de entrada das transições, a matriz de saída das transições, a capacidade de cada lugar, formalmente apresentada abaixo:

Definição 3.1 (Estrutura de Redes de Petri em Matrizes) Seja a estrutura de uma rede definida por uma quíntupla R (P, T, I O, K), onde P é um conjunto finito de lugares. T é um conjunto finito de transições. I : (P x T) $\rightarrow$ ℵ é a matriz de pré- condições. O : (P x T) $\rightarrow$ ℵ é a matriz de pós-condições. K é o vetor das capacidades associadas aos lugares (K : P $\rightarrow$ ℵ $\cup\{\Omega\}$).

Se o conjunto de lugares ou o conjunto de transições é vazio, a rede é dita degenerada.

3.3.3. Rede de Petri marcada

Uma marca (Pode também ser denominado ficha ou *token*) é um conceito primitivo em redes de Petri, tal qual lugar e transição. As marcas são informações atribuídas aos lugares. Uma marcação, ou estado, associa um inteiro k, não-negativo, a cada lugar da rede. Abaixo seguem as seguintes definições: vetor marcação; a definição de rede de Petri marcada.

Definição 3.2 (Marcação) Seja P o conjunto de lugares de uma rede R. Define-se formalmente marcação como uma função que mapeia o conjunto de lugares P a inteiros não-negativos M: P $\rightarrow$ ℵ.

Definição 3.3 (Vetor Marcação) Seja P o conjunto de lugares de uma rede R. A marcação pode ser definida como um vetor M ($M_{(p1)}$, ..., $M_{(pn)}$), onde n= #P, tal que $M_{(}$pi) $\in$ ℵ, para todo pi $\in$ P.

Definição 3.4 (Rede Marcada) Define-se uma rede de Petri marcada pela dupla RM (R; M0), onde R é a estrutura da rede e M_0 é a marcação inicial.

O comportamento dos sistemas pode ser descrito em função dos seus estados e suas alterações. Para simular o comportamento dos sistemas, a

marcação da rede de Petri é modificada a cada ação realizada (transição disparada), segundo regras de execução.

O disparo de transições (execução das ações) é controlado pelo número e distribuição de marcas nos lugares. Uma transição está habilitada se cada um dos seus lugares de entrada possuírem um número de *tokens* pelo menos igual ao peso do arco que os liga. Denota-se a habilitação de uma transição t para uma marcação Mk por Mk[t>.

Definição 3.5 (Regra de Habilitação) Seja RM (R; M_0) uma rede de Petri marcada, t$\in$ T uma transição e Mk uma marcação. Se Mk[t >, Mk(pi) $\geq$ I(pi,t), $\forall$pi $\in$ P. Formalmente, apresentamos as regras de execução das redes de Petri a seguir:

Definição 3.6 (Regra de Disparo) Seja RM = (R; M_0) uma rede de Petri marcada, t $\in$ T uma transição e uma marcação M. A transição t pode disparar quando ela está habilitada. Disparando uma transição habilitada, a marcação resultante é M =M_0 - I(pj,t) + O(pj,t), $\forall$p $\in$ P. Se uma marcação M é alcançada por M_0 pelo disparo de uma transição t, ela é denotada por M_0[t> M.

3.3.4. Matriz de incidência

A matriz de incidência representa a estrutura da rede em termos do peso de todos os arcos. Tal definição é de suma importância desde que esta estrutura seja a base para uma análise estrutural e para descrição do comportamento da rede em termos da equação de estado.

Uma maneira prática de representar a matriz de uma rede é através de pré e pós-condições de incidência. Estas matrizes representam as interconexões entre lugares e transições e também seus pesos. Quando uma transição t dispara, a diferença entre a nova marcação alcançável e uma anterior é obtida por O(pi , t) — I(pi , t), considerando cada lugar pi.

A matriz C = O - I é chamada matriz de incidência. Esta matriz representa a estrutura da rede do sistema ora modelado. Abaixo, segue uma definição formal de uma matriz de incidência:

Definição 3.7 (Matriz de Incidência) Seja R = (P, T, 1, O, K) ser uma rede de Petri.

A matriz de incidência C representa a relaçao (P x T) $\rightarrow$ Z definida por C(p, t) = O(p,t) - I(p,t),$\forall p \in P, \forall t \in T$.

O grafo das marcações acessíveis (ou alcançáveis) é uma representação gráfica do conjutno das marcações que podem ser alcançadas para uma dada rede de Petri. O disparo de uma transição modifica a marcação, conforme a marcação atual e a estrutura da rede. Essas marcações obtidas após os disparos das transições são marcações acessíveis de uma rede para uma determinada marcação inicial.

Definição 3.8 (Conjunto de Marcações Acessíveis) Seja uma rede marcada RM (R; M_0); define-se conjunto das marcações acessíveis CA(R; M0) pelo conjunto demarcações obtidas a partir de uma marcação inicial M_0 e pelo disparo de todas as possíveis seqüências de transições habilitadas, ou seja, CA(R; M_0)= { M i $\in \aleph$ m}, tal que existe uma seqüência de transições sq, que M0 [sq > Mi.

O grafo das marcações acessíveis pode ser definido por um par GA(R; M_0)= (CA(R; A),), onde CA(R; M0) são os vértices do grafo e A os arcos. Os arcos representam a alteração da marcação pelo disparo das transições. Sempre que o contexto esteja claro, utiliza-se CA ao invés de CA(R; M_0).

Definição 3.9 (Grafo das Marcações Acessíveis) Seja uma rede marcada RM = (R;M0); define-se grafo das marcações acessíveis GA, por um par GA(R; M_0) (CA, A(i,j)), onde Mi$\in$ CA(R; M_0) são os vértices do grafo e A os arcos. Os arcos representam M'[> M" se, e somente se, existe uma transição t tal que M'[t > M" (mudança de marcação).

Figura 3-4 – Abstração x Interpretação

3.4. REDES DE PETRI E SUAS CLASSES

Dois aspectos ortogonais associados às redes de Petri que devem ser ressaltados são: o nível de abstração e as diferentes interpretações. Estas características especificam um espaço de formalismos apropriados para diferentes propósitos e são apresentados na Figura 3.4. O primeiro nível de abstração são as redes elementares, onde lugares representam condições booleanas. No segundo nível estão as redes *Place-Transition*, onde lugares são variáveis onde seu domínio são inteiros não negativos. Adicionalmente algumas extensões tem sido consideradas, tais quais arcos inibidores e redes com prioridades. Tais extensões afetam não somente a concisão, mas também o poder de modelagem dos modelos adotados. A interpretação mostrada basicamente no eixo x modela em redes de Petri várias interpretações temporizadas, incluindo modelos determinísticos, disparo no tempo representado por intervalos, modelos estocásticos etc., atenuadas para análises de desempenho, escalonamento, controle de tempo real, etc.

3.5. PROPRIEDADES DAS REDES DE PETRI

Em paralelo aos modelos apresentados foi desenvolvida uma série de métodos que permitem a análise de um grande número de propriedades em sistemas [3]. Divide-se em dois tipos as propriedades das redes de Petri: as propriedades dependentes da marcação e as independentes da marcação, denominadas propriedades comportamentais e estruturais, respectivamente.

3.5.1. Propriedades Comportamentais

Entre as propriedades comportamentais [3], pode-se destacar: alcançabilidade, limitação, vivacidade, reversabilidade, segurança.

Alcançabilidade - Alcançabilidade (*reachability)* é de suma importância para o estudo de propriedades dinâmicas dos sistemas. A alcançabilidade indica a possibilidade de atingirmos uma determinada marcação pelo disparo de um número finito de transições, a partir de uma dada marcação. Será apresentado adiante este conceito no contexto de redes de Petri [3]. Dada uma determinada rede de Petri marcada RM = (R; M_0), representada na Figura 3.5, o disparo de uma transição t altera a marcação da rede, conforme as regras descritas a seguir. Uma marcação M' é dita acessível de M0 se existe uma seqüência de transições que, disparadas, levam a marcação M'. Ou seja, se a marcação M_0 habilita a transição t0, disparando-se esta transição atinge-se a marcação M_1. A marcação M_1 habilita t1 a qual sendo disparada atinge-se a marcação M2 e assim por diante até a obtenção da marcação M'. Mais formalmente tem-se:

Definição 3.11 (Alcançabilidade) Seja Mi[tj > Mk e Mk[th > M1 então Mi[tjth > M1. Por recorrência designamos o disparo de uma seqüência $s \in T^*$ por M[s > M'. O conjunto de todas as possíveis marcações obtidas a partir da marcação M_0 na rede RM = (R; M_0) é denotado por CA(R; M_0) = {M' $\in \aleph^m | \exists s$, Mo[s > M'}, onde m é a cardinalidade do conjunto de lugares da rede.

A análise da alcançabilidade de uma marcação consiste em determinarmos se uma dada marcação M'$\in$ CA(R; M_0) da rede marcada RM. Em alguns casos,

deseja-se observar apenas alguns lugares específicos da rede em estudo. Este problema é denominado sub-marcação alcançável (*submarking reachability*).

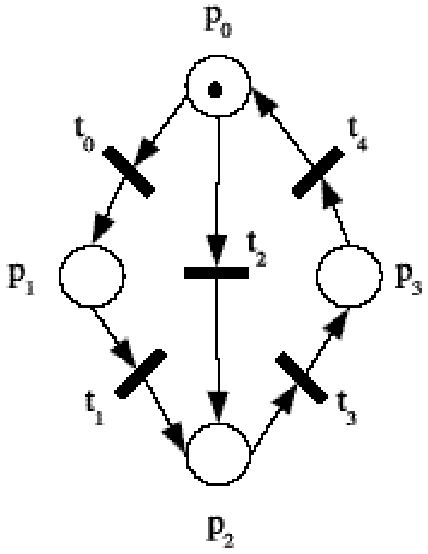


Figura 3-5 – Rede de Petri Marcada.

Muitos outros problemas de análise podem ser observados em termos do problema de alcançabilidade. Por exemplo, se uma rede fica em *deadlock* em uma determinada marcação, pode-se querer saber se essa marcação é acessível.

Limitação - Abaixo segue definição formal do conceito de limitação (*boundedness*) nas redes de Petri e sua importância na verificação de uma especificação de sistemas.

Definição 3.12 (Limitação) Seja um lugar pi $\in$ P, de uma rede de Petri marcada RM = (R; M_0). Este lugar é dito k-limitado (k-*bounded*) (k $\in$ $\aleph$) ou simplesmente limitados se para toda marcação acessível M $\in$ CA(R; M_0), M(pi) $\leq$ k.

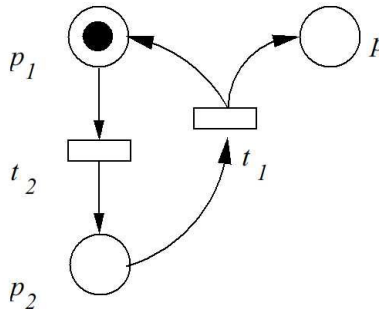


Figura 3-6 – Exemplo de rede não limitada.

Caso esta propriedade não seja identificada, o lugar é dito não-limitado (ver Figura 3.6). Perceba que o limite k é o número de marcas que um lugar pode acumular. Uma rede de Petri marcada RM = (R; M_0) é dita k-limitada se o número de marcas de cada lugar de RM não excede k em qualquer marcação acessível de RM($M(p) \leq k, \forall M \in GA(R; M_0), \forall p \in P$).

Definição 3.13 (Rede Limitada) Diz-se que uma rede RM = (R; M_0) é limitada (*bounded*) se $k(pi) \leq \infty , \forall p \in P$.

Segurança - O conceito de segurança (*safeness*) é uma particularização do conceito de limitação. Um lugar pi é dito k-limitado se o número de marcas que este lugar pode acumular estiver limitado ao número k. Um lugar que é 1-limitado pode ser simplesmente chamado de seguro (*safe*).

Definição 3.14 (Lugar Seguro) Seja um lugar $pi \in P$ de uma rede marcada RM = (R; M_0), onde pi é seguro se para toda marcação $M' \in CA(R; M_0)$ tivermos $M(pi) \leq 1$. Dizemos que a rede RM (R; M_0) é segura se todos os lugares dessa rede são seguros, ou seja, todos os lugares desta rede podem conter uma ou nenhuma marca (ver Figura 3.7).

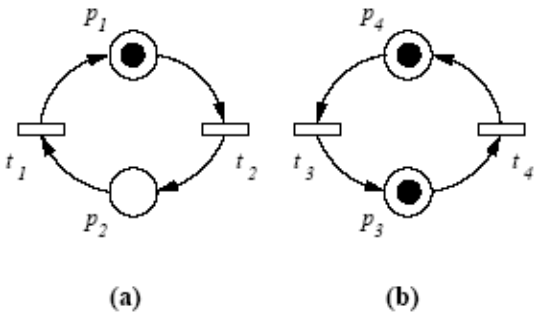


Figura 3-7 – Rede segura depende da marcação inicial.

Definição 3.15 (Rede Segura) Uma rede RM (R; M_0) é definida como segura se M(pi) ≤ 1, para todo p ∈ P.

Vivacidade (*liveness*) - Denominamos uma transição ti viva (*live*) em uma marcação M se ti é potencialmente disparável para todas as marcações M ∈ A(R; M0).

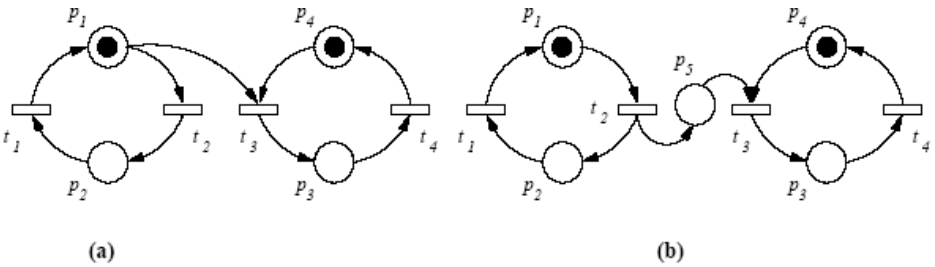


Figura 3-8 – Vivacidade.

Definição 3.17 (Rede viva) - Uma rede RM = (R; M0) é dita viva (*live*) se para toda M Î CA(R; M0) é possível disparar-se qualquer transição de RM através do disparo de alguma seqüência de transições (ver Figura 3.8).

Um bloqueio (ou *deadlock*) - em uma RdP ocorre quando se atinge uma marcação a partir da qual o disparo de uma ou mais transições não é mais possível. Portanto, a *liveness* de uma RdP indica a ausência de bloqueios.Contudo, a ausência de bloqueio não implica em *liveness*.

Reversibilidade - Uma rede de Petri RM = (R; M0) é definida como reversível se for sempre possível retornar à marcação inicial M_0, isto é, M_0 é alcançável de qualquer outra marcação acessível M.

Definição 3.18 (Reversibilidade) Uma rede RM = (R; M_0) é dita reversível se para toda e qualquer M Î CA(R; M_0) é possível retornar a marcação inicial da rede (M0) através do disparo de alguma seqüência de transições.

3.5.2. Propriedades Estruturais

As propriedades estruturais são aquelas que refletem características independentes da marcação. Desde que as redes sejam puras, a estrutura da rede pode ser representada pela matriz de incidência. Serão visto os conceitos de limitação estrutural, conservação e consistência.

Limitação Estrutural Uma rede de Petri R = (P, T, 1, O, K) é classificada como estruturalmente limitada (*structural bounded*) se é limitada para qualquer marcação inicial.

Conservação - A conservação é uma importante propriedade das redes de Petri permitindo, por exemplo, a verificação da não-destruição de recursos através da simples conservação de marcas [5]. Uma rede é dita conservativa se esta é conservativa para algum vetor de peso de inteiros positivos.

Definição 3.19 (Rede Conservativa) Uma rede marcada RM = (R; M_0) é dita conservativa se existe um vetor de pesos W = (wi, w2, ..., wn), tal que $\sum wi.Mk(pi) = \sum wi.M_0(pi)$, onde n = #P e wi é um inteiro positivo, $\forall pi \in P$ e $\forall Mk \in$ CA(R; M_0).

Consistência Uma rede de Petri tem a propriedade de consistência se existe uma seqüência de transições habilitadas a partir de uma marcação onde todas as transições da rede são disparadas pelo menos uma vez.

Definição 3.20 (Consistência) Seja RM = (R; M_0) uma rede marcada. RM é dita consistente se Mi [s > Mj e pra toda transição ti dispara ao menos uma vez em s (s(ti)>0 – número de disparos de ti em s).

Uma descrição mais detalhada destas e de outras propriedades das RdPs pode ser encontrada em [3].

3.6. MÉTODO DE ANÁLISE E VALIDAÇÃO DE RDP

Uma grande vantagem na utilização de RdPs na modelagem de sistemas é a sua capacidade de formalização para a análise e verificação das propriedades do sistema modelado. A análise destas propriedades é implementada através de três métodos [10]: geração do grafo de alcançabilidade, análise da equação de estados e técnicas de redução. A validação das propriedades pode ser realizada através da simulação.

Geração da árvore ou grafo de alcançabilidade – envolve a listagem de todas as marcações alcançáveis a partir da marcação inicial M_0. Cada nó da árvore representa uma marcação alcançável e cada arco representa o disparo de uma transição. Esse método de análise que se baseia na construção de uma árvore que possibilite a representação de todas as possíveis marcações de uma rede. A árvore de cobertura é utilizada para representar de forma finita um número infinito de marcações. Para uma dada RdP com uma marcação inicial, é possível obtermos diversas marcações para um grande número de transições potencialmente habilitadas [31]. Essas marcações podem ser representadas por uma árvore, onde os nós são as marcações e os arcos as transições disparadas. Algumas propriedades, tais como limitação, segurança, transições mortas e alcançabilidade de marcações [3], podem ser analisadas através da árvore de cobertura.

Análise através da Equação de Estado – utiliza a matriz de incidência, que representa as conexões entre lugares e transições, para representar a estrutura da rede e caracterizar o comportamento dinâmico do sistema. Esse comportamento pode ser descrito por equações diferenciais ou equações algébricas. A vantagem das técnicas algébricas sobre as técnicas baseadas na análise das árvores de cobertura é que a análise de propriedades pode ser efetuada pela resolução de equações lineares simples. As equações desenvolvidas governam o comportamento concorrente dos sistemas modelado por RdPs. Todavia, a solução destas equações é limitada, em parte devido à natureza não determinística dos modelos de RdP e por causa da restrição que soluções devem ser encontradas como inteiros não-negativos [3].

Técnicas de redução – consistem em transformações que reduzem a dimensão do grafo de alcançabilidade, mas que asseguram parcialmente a conservação das propriedades a serem analisadas. A análise de sistemas de grandes dimensões normalmente é uma operação custosa. Reduções são transformações aplicadas ao modelo de um sistema com o objetivo de simplificá-lo, e ainda preservando as propriedades do sistema a ser analisado. De modo oposto, existem técnicas para transformar um modelo abstrato em um modelo mais refinado que podem ser utilizadas para processos de síntese. Existem várias técnicas de transformação para RdPs. Aqui serão citadas apenas as mais simples, que podem ser utilizadas para a análise de vivacidade, segurança e limitação por preservarem tais propriedades. Segundo Murata [3], a maior fraqueza das RdPs é o problema da complexidade, ou seja, modelos de RdP tendem a ser muito grandes para a análise até mesmo de sistemas de tamanho reduzido. Portanto, é muito importante desenvolver métodos de transformação que permitam reduções hierárquicas ou graduais e preservem as propriedades do sistema a serem analisadas.

Simulação - utilizada quando o sistema é relativamente complexo e sua análise através de métodos analíticos se mostra inviável. Neste contexto, a simulação é utilizada para validação virtual do sistema, ou seja, para que se tenha uma maior confiança de que o sistema modelado apresenta as propriedades desejadas.

3.7. REDES DE PETRI ESTOCÁSTICAS GENERALIZADAS (GSPN)

Várias propostas para incorporação de informação temporal nos modelos de RdPs aparecem na literatura [24], [16]. Associando aos modelos RdP (estado/evento), onde o tempo é naturalmente associado com atividades que induzem mudanças de estados (eventos). Essas propostas possibilitaram a ligação das RdPs com área de avaliação de desempenho, tradicionalmente baseada em modelos estocásticos chamados de Redes de Petri Estocástica (SPN) [7] [10]. As Petri estocásticas (SPN) são introduzidas em 1980 como formalismo para descrição de sistemas de eventos discretos, onde o

comportamento dinâmico seria representado pela CTMC. As SPN compreende as transições temporizadas, as quais são associados retardos (tempos) exponencialmente distribuídos, já a extensão desses modelos estocásticos seriam as redes de petri estocástica generalizada (GSPN) compreendem dois tipos de transições: as transições temporizadas, como numa SPN; e as transições imediatas, que disparam com tempo zero, com prioridade superior a transições temporizadas. Níveis diferentes de prioridade podem ser atribuídos às transições. As prioridades podem servir para solucionar situações de confusão. Associam-se pesos às transições imediatas, a fim de solucionar situações de conflito. Outras distribuições, além da distribuição exponencial, podem ser modeladas utilizando-se o recurso de aproximação por fases de exponenciais, que será apresentada na seção 3.9.

Uma GSPN pode ser definida como uma óctupla: GSPN = (P, T, π, I , O, H, M_0, W), onde (P, T, I , O, H, M_0) é RdP não temporizada subjacente, que compreende: P um conjunto de lugar; T um conjunto de transições; I, O, H: T→ IN as funções de entrada, saída e inibidoras; M_0 uma marcação inicial; Adicionalmente, a definição GSPN compreende a função de prioridade π: T→ IN a qual associa a menor prioridade (0) às transições temporizadas e prioridades mais altas (≥1) às transições imediatas:

$$\pi \quad (t) = \begin{cases} 0 \text{ se t é temporizada} \\ \geq 1 \text{ se t é imediata} \end{cases}$$

Finalmente, o último item da definição GSPN é a função W: T→ IR, que associa um valor real às transições. Onde w(t) é o parâmetro da f.d.p. exponencial negativa das transições temporizadas e um peso usado para o cálculo das probabilidades de disparo das transições imediatas (se é uma transição imediata). Na representação gráfica da GSPN, transições imediatas são traçadas como segmentos e transições exponenciais como caixas retangulares exponenciais.

No modelo GSPN há dois tipos de estados, chamadas de estados tangíveis (*tangible*) e os estados voláteis (*vanish*). Os estados voláteis são assim denominados, porque o seu tempo de vida é igual a zero. O estado volátil é criado

em decorrência da marcação dos lugares que são pré-condição de uma transição imediata. Desta forma, quando as marcas chegam a estes lugares, são instantaneamente consumidas. O tempo de permanência das marcas nestes lugares é zero. Esta é a razão de chamá-los de estados voláteis, pois são criados e instantaneamente destruídos. Uma marcação pode ser classificada como: tangíveis e *vanishing*. Marcações com transições imediatas habilitadas são classificadas como *vanishing*. As demais são denominadas tangíveis.

O modelo GSPN faz uso da semântica *interleaving* de ações. Assume-se que as transições são disparadas uma a uma, mesmo se o estado compreenda transições imediatas não conflitantes. A análise de um modelo GSPN requer a solução de um sistema de equações igual o número de marcações tangíveis. O gerador infinitesimal Q da CTMC associado ao modelo GSPN é derivado de uma redução de um grafo de alcançabilidade, rotulado com as taxas ou pesos das transições imediatas.

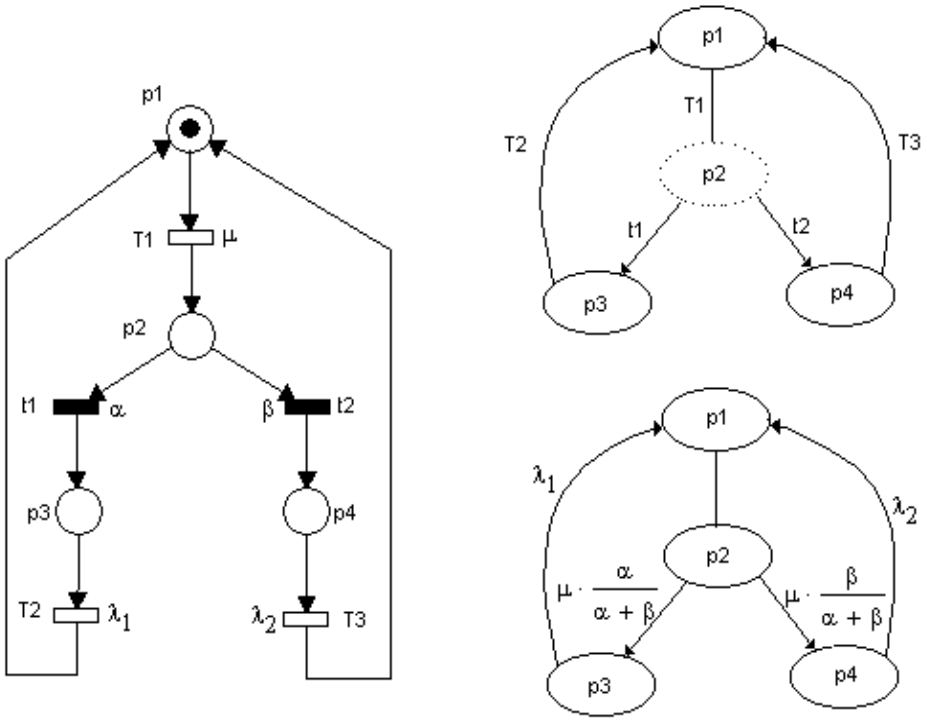


Figura 3-9 - Geração do gráfico de alcançabilidade GSPN.

A Figura 3.9 apresenta um exemplo de geração do grafo de alcançabilidade de uma GSPN. Na GSPN da Figura 3.9, à esquerda, existe um conflito entre as duas transições imediatas. Mostra-se o grafo de alcançabilidade na parte superior direita. As linhas pontilhadas do estado p2 indicam que o estado é volátil. De fato, quando a transição T1 dispara o sistema entra estado p2, habilitando as duas transições imediatas, t_1 e t_2, gerando o estado p_2. Há uma mudança imediata (tempo zero) para o estado p3 ou p4, através do disparo da transição t_1 ou t_2, com probabilidades $\frac{\alpha}{\alpha+\beta}$ e $\frac{\beta}{\alpha+\beta}$ respectivamente. O grafo de alcançabilidade tangível na parte direita inferior de Figura 3.9 é obtido eliminando-se o estado volátil p_2. A taxa na qual o sistema se move do estado p1 para p_3 ou p_4 é obtida pelo produto da taxa λ da transição do estado p_1 para o estado volátil p_2, com a probabilidade de ir do estado p_2 para o estado p_3 ou p_4.

Desta forma, [3] chegou ao seguinte teorema: “qualquer rede de Petri estocástica marcada, com um número finito de lugares e transições, é isomórfica a uma cadeia de Markov”.

De acordo com [19] e [22], obtêm-se o isomorfismo da GSPN com uma cadeia de Markov a partir do grafo de alcançabilidade reduzido, que é dado através da eliminação dos estados voláteis (não tangíveis) e com as taxas de transição entre os estados rotulando os arcos. A avaliação se dá através da cadeia de Markov embutida na GSPN. Podem ser realizadas avaliações em estado estacionário, estado transiente ou através de simulação.

A solução em estado estacionário e transiente da GSPN é obtida pela solução da CTMC subjacente, o qual foi mostrada na sessão de cadeias de Markov de tempo contínuo. Portanto, a rede de Petri é o método formal utilizado para modelagem neste trabalho e os modelos obtidos serão passíveis de verificações, análises e avaliações.

3.8. REDES DE PETRI ESTOCÁSTICA DETERMINÍSTICA (DSPN)

Nessa seção, introduzem-se as Redes de Petri Estocástica Determinísticas (DSPN). Nas DSPN, além das transições definidas pelas GSPN, acrescentam-se transições determinísticas para representação do retardo (tempo). As DSPN são definidas por uma tupla: DSPN = (P, T, π, I , O, H, M_0, D,W). Onde (P, T, I , O, H, M_0) é a RdP não temporizada subjacente, que compreende: P um conjunto de lugar; T um conjunto de transições; I, O, H: T→ IN as funções de entrada, saída e inibidoras, respectivamente; M_0 uma marcação inicial. Adicionalmente, a definição DSPN compreende a função de prioridade π: T→ IN, à qual associa a menor prioridade (0) às transições temporizadas e prioridades mais altas (≥1) às transições imediatas:

$$\pi \quad (t) = \begin{cases} 0 \text{ se t é temporizada} \\ \geq 1 \text{ se t é imediata} \end{cases}$$

O termo que difere da definição da GSPN é o termo D, que representa o tempo de disparo associado às transições temporizadas DSPN. Isso especifica o tempo de disparo de cada transição exponencial. Finalmente, o último item da definição DSPN é a função W: Tà IR, que associa um valor real às transições.

Onde w(t), é o parâmetro da f.d.p. exponencial negativa das transições temporizadas e um peso usado para o cálculo das probabilidades de disparo das transições imediatas (se é uma transição imediata). Nota-se que o formalismo introduzido inclui a definição de GSPN, pois algumas GSPN podem ser definidas como uma DSPN sem transições determinísticas.

3.9. DISTRIBUIÇÃO POR FASE

As GSPN modelam ações imediatas e exponenciais. Há uma preocupação que essa restrição limite à precisão dos resultados de desempenho, quando o sistema contenha transição de estados não exponenciais. As cadeias Semi-Markovianas fornecem uma estrutura matemática simples para inclusão de distribuições gerais na estrutura do modelo Markoviano. Foi criado o conceito de construtores GSPN chamados de *throughput subnet* e *s-transitions*. Esses dois construtores ampliam os tipos de distribuição de transição de estados que pode ser precisamente modelada com uma GSPN. Uma variedade de formas de *delay* pode ser construída no modelo GSPN, usando-se os construtores *throughput subnet* e *s-transitions*.

Sabe-se que o modelo GSPN mostra precisamente as distribuições de tempo exponenciais, porém, para aumentar a abrangência do modelo para outros tipos de distribuição não-exponencial, utilizou-se o conceito de construtores GSPN *throughput subnet* e *s-transitions* para modelar distribuições não-exponenciais.

Várias combinações de lugares, transições exponenciais e transições imediatas podem ser usadas entre dois lugares para guardar diferentes tipos de distribuição.

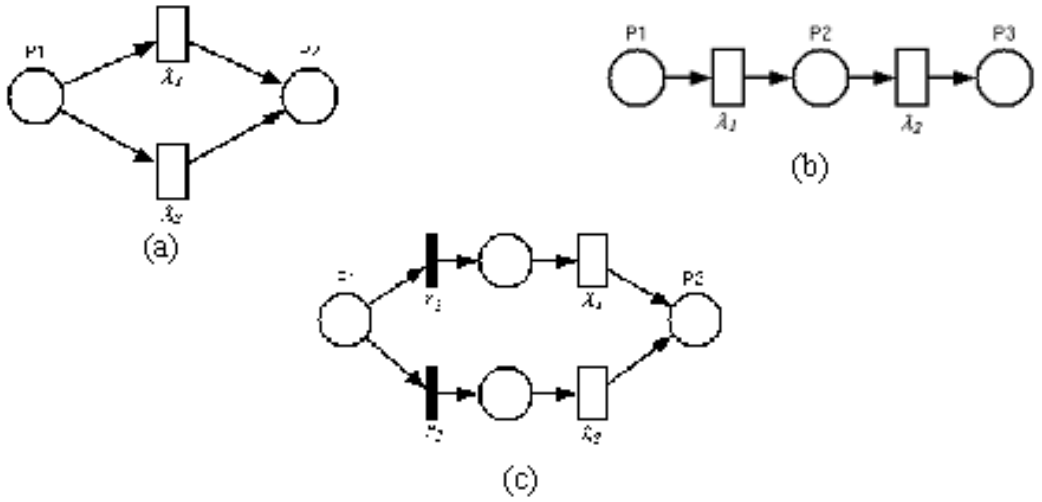


Figura 3-10 - Diferentes tipos de distribuições.

A Figura 3.10 descreve três *throughput subnet* formados por conexões série e paralelo. A Figura 3.10a consiste em duas transições exponenciais em paralelo, com taxa de parâmetro λ_1 e λ_2, respectivamente. Um *token* aparecendo em p_1 aparecerá em p_2 depois de dois curtos *delays* r_1+ r_2, tem decorrido. O resultado da função de densidade com esse *delay* é dado por:

$$r = \min(r_1 + r_2)$$
$$f_r(t) = (\lambda_1 + \lambda_2)? \; ^{(\lambda 1 + \lambda 2)t}, t > 0.$$

Assim, esse tipo de interconexão é equivalente para uma transição exponencial com parâmetro $\lambda_1 + \lambda_2$ e, conseqüentemente, não fornece adicional flexibilidade de modelagem com respeito análise de desempenho.

A Figura 3.10b consiste em duas transições exponenciais em série com parâmetro $\lambda_1+\lambda_2$, respectivamente. O *delay* resultante é $t = t_1+t_2$, resultando na função de densidade abaixo:

$$f_{r(t)} = (f_{r_1} * f_{r_2})(t)$$

$$= \lambda_1\lambda_2 \frac{(e^{-\lambda_1 t} - e^{-\lambda_2 t})}{(\lambda_2 - \lambda_1)}, t > 0$$

Onde * é o operador convolução. Esta expressão generaliza para mais que duas transições. Para o caso onde $\lambda_1 = \lambda_2 = ... = \lambda_n$, a função densidade vem:

$$f_r(t) = \frac{\lambda^n}{(n-1)!} t^{n-1} e^{-\lambda t}, t > 0$$

Isto é, uma distribuição do tipo *Erlang* de ordem N. Em particular, a distribuição do tipo *Erlang* é especificada por dois parâmetros $\lambda > 0$ e $n > 0$.

A Figura 3.10c consiste em uma distribuição hiper-exponencial que é modelada com duas ramificações paralelas, cada uma contendo uma transição imediata e outra exponencial. A transição imediata forma a probabilidade de mudar do lugar p_1. Quando um *token* chegar em p_1, a probabilidade de cada ramificação é determinada pelos pesos das transições r_1 e r_2. A função de densidade é:

$$f_\tau(t) = r_1 f_\tau(t) + r2 f_\tau(t) = r_1\lambda_1 e^{-\lambda 1 t} + r_2\lambda_2 e^{-\lambda 2 t}, t > 0$$

Esta *throughput sub-net* implementada é uma função de *delay* hiper-exponencial. Uma particular distribuição exponencial é caracterizada pelo seguinte:

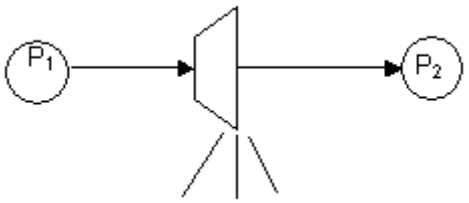
n, a ordem
r_j, =1...n, $\sum r_j = 1$
λ_i, j=1…n.

Esta *throughput sub-net* descrito acima são usados para implementar distribuições Erlang e hiper-exponencial.

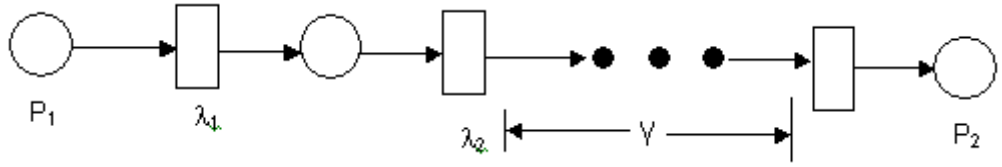
Outra regra é exatamente implementar uma distribuição com transição arbitrária com um modelo GSPN. O método usado aqui é *moment matching* [15], onde o resultado do *throughput sub-net* é chamado de *s-transition*. Posteriormente será mostrado o algoritmo que traduz o primeiro e segundo momento de uma distribuição.

Assume-se que a transição de *delay* D tem função de densidade, com tempo médio (μ_D) e desvio padrão (σ_D). A Figura 3.9 mostra o modelo para uma

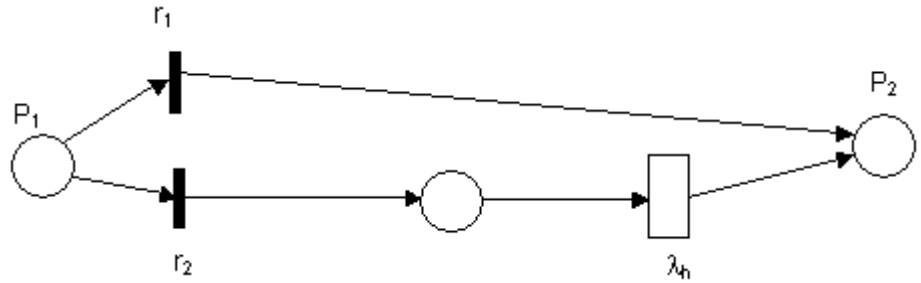
s-transition que cria esse *delay* entre dois lugares p_1 e p_2. Em alguns casos, a *s-transition* é implementada com uma distribuição exponencial em série; com uma sub-rede Erlang; em outro caso, uma sub-rede hiper-exponencial é usada. A implementação usada depende da $f_D(.)$. Os parâmetro (λ_1, λ_2 e γ), em um caso, ou (r_1 , r_2 e λ_h, em outro caso) são escolhidos, assim, a média e o desvio padrão do *throughput sub-net* são μ_D e σ_D, respectivamente.



Modelo para do Caso 1 ao 3:



Modelo para o Caso 4:



Algoritmo para o primeiro e segundo momento:

Considere os quatro casos seguintes, onde o primeiro refere-se aos três caso para um modelo *s-transition* em série, e o quarto caso refere-se a um modelo *s-transition* em paralelo.

Caso 1: Se $\mu_D = \sigma_D$, então apenas uma transição exponencial é necessária. Coloca-se γ para zero(0), e λ_1 para $1/ \mu_D$.

Caso 2: Se μ_D / σ_D é um numero inteiro(X), e X # 1, então apenas uma sub-rede Erlang é necessária (isto é $\lambda_1 = \infty$). Coloca-se γ para X^2, e $\lambda_2 = X^2/ \mu_D$.

Caso 3: Para toda $\mu_D > \sigma_D$,ambos, um simples transição e a sub-rEde ErLang são necessárias. Derivações de γ , λ_1 e λ_2 conduz para um único γ,

$$\left(\frac{\mu_D}{\sigma_D}\right)^2 - 1 \leq \gamma \leq \left(\frac{\mu_D}{\sigma_D}\right)^2$$

$$\lambda_1 = \frac{1}{\mu_1} \text{ onde } \mu_1 = \frac{\mu_{D \pm \sqrt{\gamma(\gamma+1)\sigma^2 - \gamma\mu_D^2}}}{\gamma+1}$$

$$\lambda_2 = \frac{1}{\mu_2} \text{ onde } \mu_2 = \frac{\gamma\mu_{D \pm \sqrt{\gamma(\gamma+1)\sigma^2 - \gamma\mu_D^2}}}{\gamma+1}$$

Onde qualquer uma das equações acima pode ser utilizada.

Caso 4: Para toda $\mu_D < \sigma_D$, a sub-rede hiper-exponencial é usada. Derivações de r_1 e r_2 λ_n conduz a:

$$r_1 = 2\,\mu^2_D / (\mu^2_D + \sigma^2_D)$$
$$r_2 = 1 - r_1$$
$$\lambda_h = 2\,\mu_D / (\mu^2_D + \sigma^2_D)$$

Esta técnica flexibiliza o conjunto de distribuições que são suportados pelo método proposto. E o caso 3 foi utilizado neste trabalho como refinamento para geração de modelos.

3.10.SIMULAÇÃO ESTOCÁSTICA DE EVENTOS DISCRETOS

Um dos problemas em analisar sistemas GSPN é o crescimento de seu espaço de estados, impossibilitando a construção do grafo de alcançabilidade. Nesses casos, torna-se também impossível construir associação de possessos estocásticos, então um caminho para analisar esses sistemas seria através de simulação. A simulação de um sistema GSPN para avaliação de desempenho

corresponde à geração de uma amostra através do espaço de estado do sistema e coleta de medidas sobre aspectos particulares do comportamento de um sistema GSPN.

Sistemas GSPN são adequados para utilização de simulação de eventos discretos [19], [21] no qual os eventos caracterizam a mudança de estados e os tempos gastos nos estados são aspectos relevantes para o comportamento do modelo. Os eventos correspondem ao disparo de transições e o tempo gasto nos estados deriva-se da distância temporal entre disparo de transições. A simulação de sistemas GSPN corresponde à identificação do evento corrente que está certo em acontecer dado à marcação corrente, bem como, os outros eventos que podem acontecer no futuro, dado a informação fornecida pela mesma marcação. Disparos de transição que correspondem ao evento corrente produzem uma mudança de marcação na qual novas transições podem torna-se habilitadas, e outras transições podem torna-se desabilitadas.

A identificação das transições que são habilitadas em uma dada marcação bem como as transições que são afetadas pela mudança de marcação devido ao disparo de transição é um dos principais problemas de uma simulação eficiente em grandes sistemas GSPN. Quando o conjunto de transições do GSPN é grande, testes em cada mudança de marcação todos os elementos do conjunto visto no qual transições habilitadas podem ser extremamente tempo consumidas. Uma vantagem considerável pode ser obtida pela exploração de relações interessantes entre transições que podem ser obtidas de uma análise estrutural do modelo.

3.11.CONSIDERAÇÕES FINAIS

RdPs constituem-se em uma poderosa família de técnicas de modelagem de sistemas concorrentes, provendo um grande conjunto de ferramentas de análise. Várias das características das RdPs as tornam particularmente úteis para a modelagem e análise de sistemas distribuídos.

Os modelos iniciais de RdPs não incluem a noção de tempo e objetivam modelar apenas o comportamento lógico do sistema, através da descrição das

relações causais existentes entre os eventos. A introdução da especificação de tempo é crucial quando são considerados problemas que envolvem a análise de desempenho, escalonamento e controle de tempo real, por exemplo. Para tanto, foram definidas várias extensões temporizadas para as RdPs.

Para sistemas distribuídos, que são usualmente de grandes dimensões, os modelos RdPs gerados apresentam um grande número de elementos, tornando o número de espaços gerados muito grande, sendo a técnica de distribuição por fase utilizada como alternativa para facilitar a modelagem dos sistemas distribuídos. Alguns tipos de RdP, tais como as RdPs estocásticas que são de particular importância por servirem de base para formalização dos modelos apresentados neste trabalho.

4. MIDDLEWARE

Este capítulo introduz os conceitos básicos de middleware, apresenta os modelos de middleware existentes, com ênfase no middleware orientado a objetos, mostra os principais conceitos de CORBA e o seu serviço de controle de concorrência.

4.1. CONCEITOS BASICOS

Uma aplicação distribuída caracteriza-se por ter suas partes componentes executando e trocando informações em vários pontos diferentes da rede. A implementação deste tipo de aplicação é uma tarefa inerentemente complexa, pois as partes executam em ambientes potencialmente heterogêneos e dependem fundamentalmente das condições da rede de comunicação. Por exemplo, uma parte da aplicação pode estar executando em um *laptop* usando o sistema operacional Windows, enquanto outra executa em um computador pessoal usando o sistema operacional Unix. Ao mesmo tempo, estas duas partes podem estar se comunicando através de uma rede sem-fio que de um momento para outro pode tornar-se lenta, insegura ou ainda passar a ter uma alta taxa de perdas.

Para facilitar o desenvolvimento das aplicações distribuídas e tratar os problemas que surgem com a distribuição, ambientes e padrões têm sido propostos e implementados desde o início da década de 90. A partir desta época, as redes passaram a fornecer uma infra-estrutura de comunicação adequada (rápida e confiável) para permitir a construção das aplicações distribuídas. Usando esta infra-estrutura, foram criados os primeiros ambientes conhecidos como plataformas de distribuição e hoje amplamente difundidos como *middleware*s:

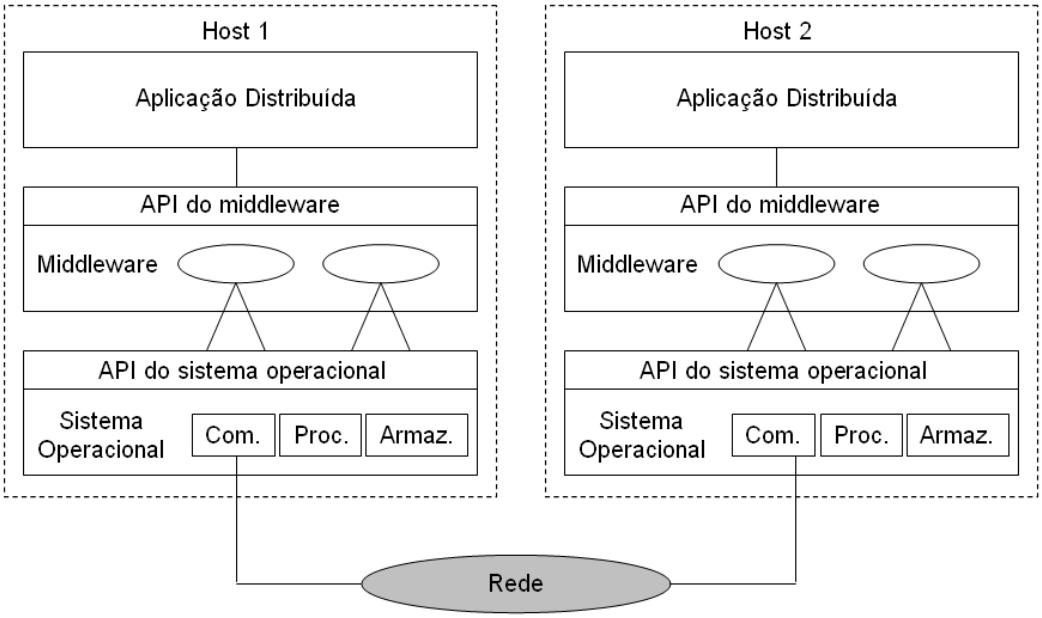


Figura 4-1 - Visão geral de *middleware*

O *middleware* é uma camada intermediária entre a aplicação e o sistema operacional (ver Figura 4.1) que permite a comunicação entre as partes da aplicação distribuída. A comunicação é feita independente da heterogeneidade de linguagens, redes, sistemas operacionais e hardware. De forma mais prática, o *middleware* fornece um conjunto de serviços de comunicação atuando como uma interface para que a aplicação seja construída sem que o desenvolvedor tenha que tratar diretamente com a complexidade da infra-estrutura de comunicação. Ao contrário dos primeiros sistemas de *middleware*, e.g., ANSAware e DCE (*Distributed Communication Environment*) , que forneciam essencialmente os serviços de comunicação e distribuição, os sistemas de *middleware* atuais, e.g., CORBA (*Commom Object Request Broker Architecture*) , DCOM (*Distributed Component Object Model*) e EJB (*Enterprise JavaBeans*) , disponibilizam serviços adicionais relacionados à segurança, comunicação assíncrona, transação, tolerância a falhas, concorrência, coordenação, nomes, páginas amarelas e outros. O conceito atual de *middleware* é bastante abrangente e tem sido utilizado em vários contextos de desenvolvimento de softwares e aplicado a diversos tipos de programas, desenvolvidos em contextos diferentes e com propósitos diferentes. O que existe em comum entre estes diversos componentes denominados de

middleware é o fato deles esconderem das aplicações detalhes de mais baixo nível dos ambientes onde elas executam.

Comum a todos os sistemas de *middleware* mencionados está o fato de que eles possuem dois componentes básicos: um conjunto de ferramentas de desenvolvimento e um ambiente de execução. O primeiro auxilia a construção da aplicação e inclui linguagens para descrição das interfaces dos serviços, conhecidas como IDLs (*Interface Definition Languages)*, e compiladores que mapeiam estas linguagens em linguagens de programação tradicionais como C, C++, Java, etc. O segundo componente do *middleware* é o conjunto de serviços que fornece o suporte necessário para a execução da aplicação. A aplicação distribuída usa estes serviços ao mesmo tempo que também disponibiliza/utiliza serviços específicos da aplicação. A utilização dos serviços disponibilizados pelo *middleware* é feita através de chamadas de operações definidas nas APIs (*Application Programming Interface*) (ver Figura 4.1). Complexos mecanismos são usados para implementar estes serviços e cujas implementações são absolutamente transparentes ao desenvolvedor da aplicação. Por exemplo, um servidor pode disponibilizar um serviço de vídeo e utilizar um serviço de comunicação multicast fornecido pelo *middleware* para enviar vídeos para diversas partes da aplicação simultaneamente.

A primitiva de comunicação disponibilizada pelo *middleware* para comunicação entre as aplicações pode ser usada para classificar os diversos modelos de *middleware* existentes atualmente. Neste caso, as primitivas de comunicação comumente implementadas incluem a chamada remota de procedimento, a invocação de método, a invocação de transação e a passagem de mensagem. Sendo assim, tem-se os modelos de *middleware* orientado a objeto (invocação de método), *middleware* orientado a mensagem (passagem de mensagem), *middleware* procedural (chamada remota de procedimento), e *middleware* transacional (invocação de transação).

4.2. MIDDLWARE ORIENTADO A OBJETOS

Middleware orientado a objetos é uma evolução do *middleware* procedural. A idéia de chamada remota de procedimento é mantida, só que com mecanismos mais sofisticados associados aos conceitos de orientação a objetos (OO). Assim, nos mecanismos de chamada de procedimento, denominado em OO de método, são contemplados os conceitos de herança e de referências, usados em linguagens de programação orientadas a objetos.

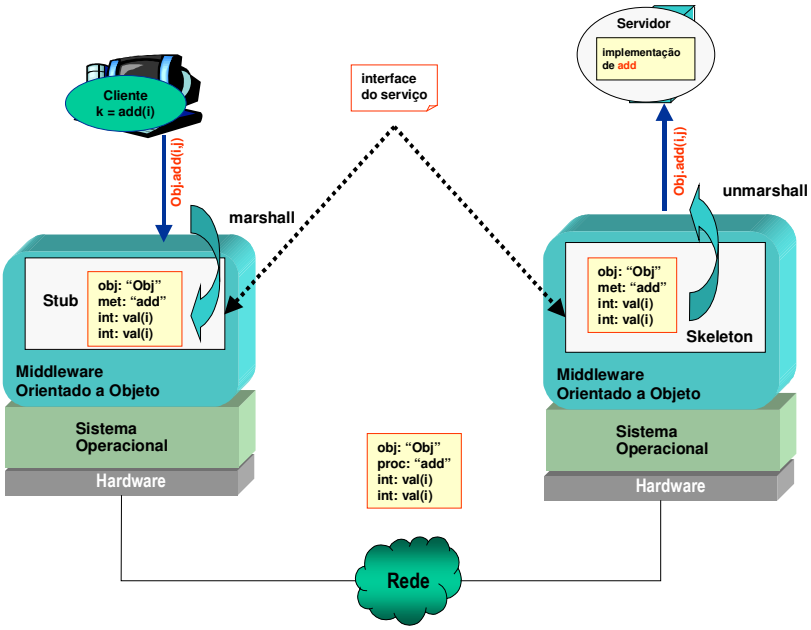


Figura 4-2 - Visão geral da comunicação em um *middleware* orientado a objetos

A Figura 4.2 mostra uma visão geral das partes componentes de um *middleware* orientado a objetos. Neste contexto surge um componente de conversão de chamadas e de objetos em dados a serem enviados pela rede – o stub. Além do stub, o skeleton aparece como o elemento que converte dados da chamadas e o retorno de objetos e de exceções em dados a serem retornados para a aplicação cliente através da rede. Neste contexto, surgem elementos responsáveis por fazer com que as aplicações possam realizar chamadas remotas

de métodos com parâmetros e retornos definidos como objetos de forma transparente e simples.

Alguns produtos e padrões incluídos na classificação de *middleware* orientado a objetos incluem o CORBA (*Common Object Request Broker Architecture*) objeto de estudo dessa dissertação, o COM (Component Object Model), o RMI (*Remote Method Invocation)* e o padrão EJB (*Enterprise JAVA Beans*).

O ponto forte dos sistemas de *middleware* orientado a objetos é que eles normalmente provêem um modelo bastante poderoso de interação remota entre objetos. No entanto, as limitações referentes à escalabilidade fazem com que o uso desta categoria de *middleware* seja restrito quando é feito em aplicações de larga escala.

4.3. CORBA

O CORBA [11] é um padrão de *middleware* orientado a objetos definido pelo OMG. O CORBA permite que aplicações distribuídas em uma rede (local ou mesmo na Internet) se comuniquem. As aplicações podem executar em diferentes plataformas de hardware (Intel, Sun, etc), sistemas operacionais distintos (Windows, Linux, Unix, etc) e podem ter sido construídas em diferentes linguagens de programação (C, C++, Java, Ada). Utilizando o padrão CORBA é possível ter aplicações completamente distribuídas, e executando em várias plataformas distintas, sem que o usuário perceba que isto está acontecendo e sem que o desenvolvedor precise se preocupar em criar soluções que resolvam os problemas de interoperabilidade entre os diferentes pedaços da aplicação.

Uma grande vantagem de CORBA é ser um padrão diretamente suportado por empresas em todo o mundo e com dezenas de implementações disponíveis, incluindo algumas gratuitas. Na prática, essa padronização significa que você não precisa ficar preso a determinados fornecedores, plataformas ou produtos, como acontece quando são escolhidas soluções proprietárias.

Tal arquitetura define um módulo intermediário ("barramento") entre clientes e servidores, o *Object Request Broker* (ORB). Neste modelo, objetos clientes

requisitam serviços às implementações de objetos (*Servants*) através de um ORB. O ORB é responsável por todos os mecanismos requeridos para encontrar o objeto, preparar a implementação de objeto para receber a requisição, e executá-la. O cliente vê a requisição de forma independente de onde o objeto está localizado, qual linguagem de programação ele foi implementado, ou qualquer outro aspecto que não está refletido na interface do objeto.

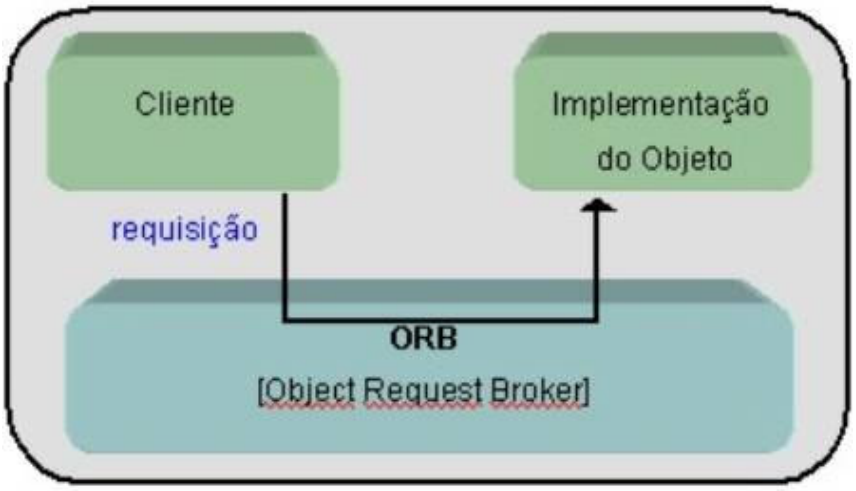


Figura 4-3 - Uma requisição sendo enviada através do ORB

Para isso, a CORBA utiliza uma IDL (*Interface Definition Language*) como uma forma de descrever interfaces. OMG IDL é uma linguagem puramente declarativa baseada em C++. Por ser uma linguagem declarativa, a IDL não possui nenhuma estrutura algorítmica ou de variáveis. São descritos em IDL apenas os tipos, as constantes e as operações necessárias para especificar uma interface de objeto. Isso garante que os componentes em CORBA sejam auto-documentáveis, permitindo que diferentes objetos, escritos em diferentes linguagens, possam interoperar através das redes e de sistemas operacionais.

Para fazer uma requisição, o cliente pode utilizar rotinas stub IDL que definem os objetos de acordo com as operações que podem ser executadas e seus parâmetros. Essas rotinas são geradas na compilação da descrição da interface do objeto ao qual se destina o pedido. Entretanto, pode-se utilizar a interface de invocação dinâmica (DII), permitindo assim, acesso aos seus serviços em tempo de execução. Sendo necessário para tanto, a inclusão da interface do

objeto ao repositório de interfaces. Um Repositório de Interfaces é uma base de dados que contém interfaces OMG IDL. Seus serviços basicamente permitem o acesso, armazenagem e atualização dessas interfaces.

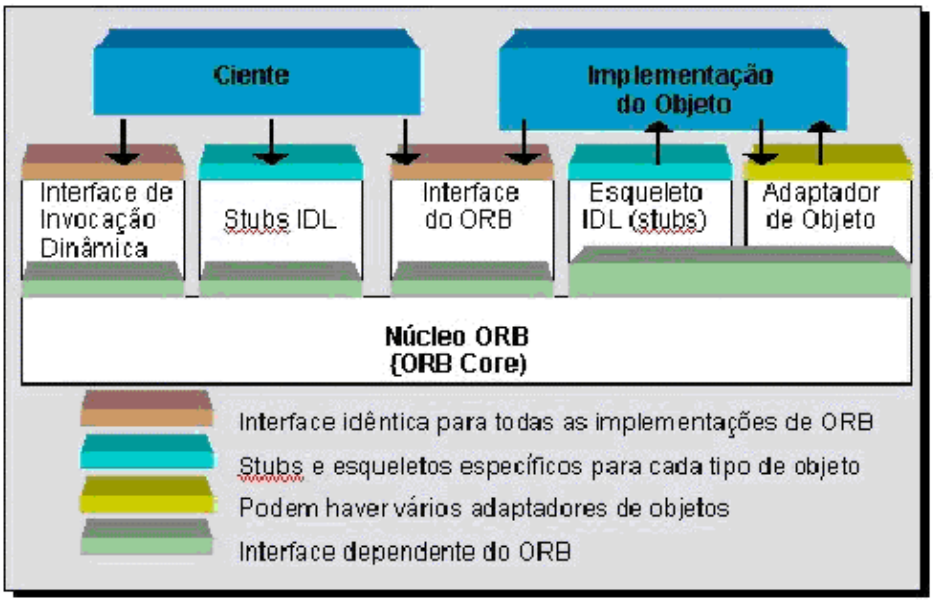


Figura 4-4 - Estrutura das Interfaces do ORB.

Tanto a invocação dinâmica (através da DII) quanto a estática (através de rotinas stub IDL) satisfazem a mesma semântica de requisição, e o receptor da mensagem não sabe dizer como o pedido foi invocado.

De forma semelhante ao que ocorre na requisição de um cliente, o ORB pode invocar a implementação de objeto através de um skeleton IDL ou de um skeleton Dinâmico (DSI - *Dynamic Skeleton Interface*). Através da DSI pode-se enviar requisições de um ORB para uma implementação de objetos que não possui informações sobre a implementação do objeto em tempo de compilação. DSI é útil em soluções de interoperabilidade implementando pontes entre diferentes ORBs.

Para executar uma requisição, a implementação do objeto pode obter alguns serviços fornecidos pelo ORB através de um Adaptador de Objetos. A implementação do objeto pode escolher qual adaptador de objetos utilizar. Essa decisão é baseada no tipo de serviço que a implementação do objeto necessita.

Para localizar e ativar implementações de objetos, um ORB se utiliza de um Repositórios de Implementações. Esses repositórios servem adicionalmente para armazenar informações adicionais associadas com implementações de ORBs, tais como, informações de depuração, controles administrativos, segurança, etc.

4.4. SERVIÇO DE CONTROLE DE CONCORRÊNCIA

O Serviço de Controle de Concorrência (SCC) de CORBA [63] foi definido para coordenar o acesso a recursos compartilhados, através do uso de *locks*, garantindo a consistência quando o recurso é acessado concorrentemente. Na prática, o SCC garante a serialização das operações realizadas concorrentemente sobre um determinado recurso. O SCC faz parte do conjunto de serviços conhecidos como serviços comuns do CORBA e são usados por várias aplicações de diversos domínios.

As noções de recurso e *lock* são básicas no SCC. Em relação à primeira, o SCC não define precisamente o que é um recurso (e nem a sua granularidade) e deixa para o desenvolvedor definir e identificar potenciais conflitos na utilização dos recursos. Por sua vez, um *lock* é utilizado para coordenar o acesso a recursos compartilhados e representa a permissão para um cliente acessar um recurso específico em um modo particular. Sendo assim, um cliente deve obter um determinado *lock* para o recurso antes de acessá-lo. Caso um outro cliente já esteja acessando-o, o SCC define potencias conflitos entre os modos de acesso dos dois clientes.

Os modos definidos pelo SCC correspondem às formas de acesso a um recurso compartilhado e são definidos como segue:

- *read*: permite a execução de uma operação de leitura sobre o recurso;
- *write*: permite a execução de uma operação de escrita sobre o recurso;
- *upgrade*: soluciona o conflito de dois clientes solicitarem a leitura de um mesmo recurso ao mesmo tempo;
- *intention read*: seleciona uma operação de leitura sobre o recurso;
- *intention write*: seleciona uma operação de escrita sobre o recurso.

Usando estes modos, um cliente somente obtém um *lock* para um recurso se o modo de *lock* requerido não conflitar com o modo utilizado por outro cliente concorrente. A Tabela 4.1 mostra a relação entre os modos:

Tabela 4.1 – Compatibilidade de *locks*

Modo Corrente	Modo Solicitado				
	IR	R	U	IW	W
Intention Read (IR)					*
Read (R)				*	*
Upgrade (U)			*	*	*
Intention Write (IW)		*	*		*
Write (W)	*	*	*	*	*

O SCC disponibiliza um conjunto de interfaces que suporta duas formas de operação: transacional e não-transacional. Quando uma aplicação utiliza a forma transacional, o *lock* obtido para o controle de concorrência é liberado pelo próprio SCC após o *commit* ou *rollback* da transação. Por outro lado, na forma não-transacional, é responsabilidade do desenvolvedor da aplicação liberar os *locks* no momento que considerar adequado.

As interfaces definidas no SCC são as seguintes: LockCoordinator, Lockset, TransactionalLockset, e LocksetFactory. A interface LockCoordinator permite a liberação de todos os *locks* mantidos por uma transação e é utilizada por clientes transacionais que precisam liberar os *locks* após o *commit* ou *rollback* da transação. A interface Lockset fornece as operações básicas para obtenção e liberação dos *locks*. A especificação desta interface no padrão é definida como segue:

```
interface LockSet {
void lock(in lock_mode mode);
boolean try_lock(in lock_mode mode);
void unlock(in lock_mode mode) raises(LockNotHeld);
void change_mode(in lock_mode held_mode, in lock_mode
new_mode) raises(LockNotHeld);
LockCoordinator get_coordinator(in CosTransactions::Coordinator
which); }
```

Figura 4-5 - Definição da interface *Locks*et

O *Locks*et contém um conjunto de *locks* que pode ser obtido para um simples recurso, como também os métodos para obter e liberar um tipo específico de *lock*. Essa interface também é responsável por assegurar que *locks* conflitantes não sejam mantidos por diferentes clientes. Todo o recurso compartilhado no sistema distribuído que requer o controle de concorrência tem seu próprio objeto de *Locks*et instanciado. Os clientes que obtém um recurso devem fazer assim somente após ter adquirido os *locks* necessários do objeto apropriado de *Locks*et.

A função *lock*, quando executada adquire um *lock* para um recurso especificado em determinado modo. Se o modo requerido for incompatível com o modo de outro cliente a operação será bloqueada e esperará o recurso ser liberado para a efetivação do *lock*. A função trylock retorna false se o *lock* requerido é incompatível com o modo de outro cliente, e true caso o *lock* possa ser adquirido. A operação *unlock* libera o recurso de *lock* no modo específico e a operação change_mode muda o modo de um simples *lock*. O controle do acesso é conseguido evitando que mais de um cliente com diferentes *locks* de acesso usem o mesmo recurso quando suas atividades forem conflitantes.

A interface TransactionalLockset é equivalente a interface Lockset, mas é utilizada para obtenção de *locks* na execução de transações. Por fim, a interface LocksetFactory possui as operações para a criação de Lockset e LockCoordinator.

Vale a pena salientar que nesta dissertação a interface utilizada foi a *Locks*et e em função disto o seu maior detalhamento nesta seção. A especificação completa das interfaces e suas operações podem ser encontradas em [11].

4.5. CONSIDERAÇÕES FINAIS

Este capítulo apresentou as características básicas do *middleware*, tendo como enfoque o *middleware* orientado a objetos, CORBA, especificamente o serviço de controle de concorrência. Esse serviço foi descrito em detalhes juntamente com suas interfaces, que serão modeladas e avaliadas no Capítulo 6 e 7, respectivamente.

5. METODOLOGIA DE AVALIAÇÃO

Este capítulo apresenta a metodologia utilizada para modelar e avaliar o desempenho de sistemas. Para efetuar uma avaliação de desempenho com qualidade, além de escolher a técnica a ser aplicada, é importante definir e seguir um procedimento sistemático que descreva todos os passos envolvidos em cada etapa do processo de avaliação.

A avaliação de desempenho dos sistemas computacionais através de modelos compreende um conjunto de atividades, tais quais: seleção de técnicas de avaliação, definição de métricas de desempenho, compreensão do sistema avaliado etc. Nesse processo, cada sistema avaliado precisa ser bem compreendido para que seja modelado, assim como também a seleção cuidadosa dos métodos de avaliação.

Para que se execute, com sucesso, a avaliação de desempenho utilizando-se modelos alguns aspectos devem ser considerados, como a complexidade dos modelos e a adequação de recursos computacionais disponíveis. A metodologia proposta possui fases bem definidas, que contemplam a modelagem e o processo de avaliação. Todas as fases dessa metodologia serão descritas nas seções posteriores e sua aplicabilidade para o desenvolvimento deste trabalho será apresentada em detalhes nos Capítulos 6 e 7.

5.1. VISÃO GERAL DA METODOLOGIA

Essa metodologia é composta por várias etapas que compreendem as atividades que vão desde a compreensão do sistema a ser avaliado até a apresentação dos resultados alcançados. Apesar das métricas, as cargas de trabalho e as técnicas de avaliação de desempenho escolhidas variarem quando se avalia um sistema específico, existe um conjunto de passos comuns envolvidos (ver Figura 5.1) nesse processo. O processo é descrito através dos diagramas de

atividades [20], os quais têm semântica muito próxima das redes de Petri *place/transition*.

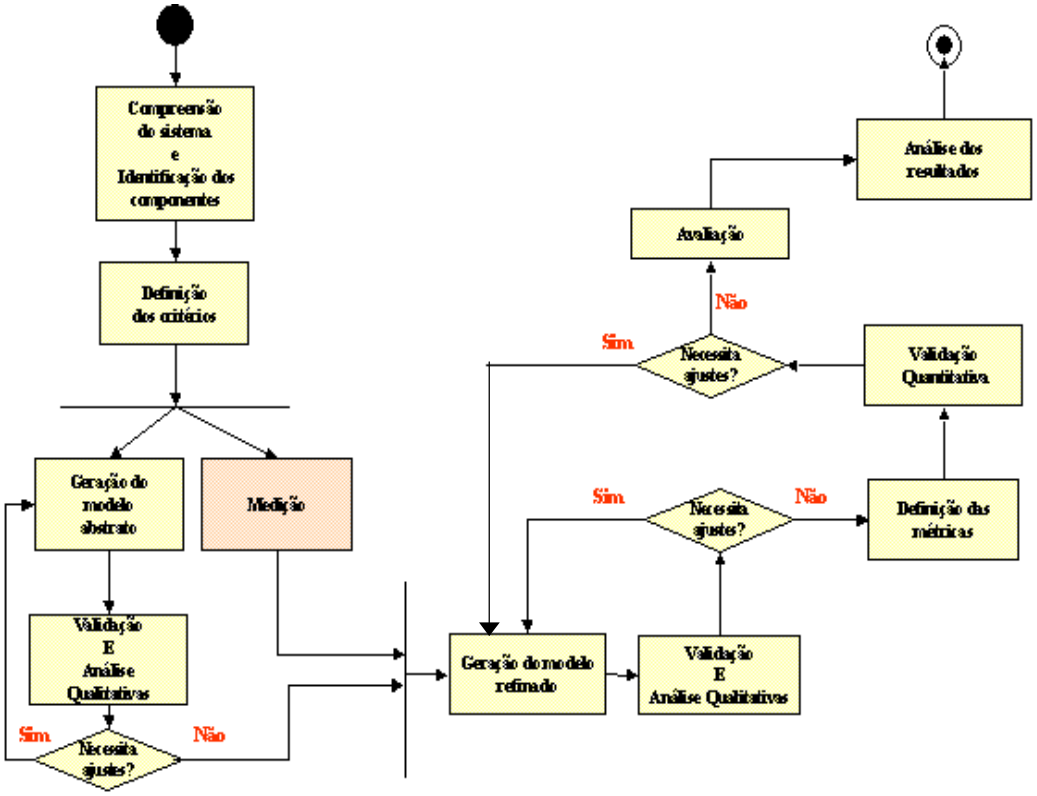


Figura 5-1 – Diagramas de atividades.

Na fase de compreensão do sistema a ser avaliado, o ajuste do problema (sistema) a ser avaliado é estudado de maneira a definir os componentes relevantes e suas iterações internas com o ambiente. Nesta fase devem ser identificados os componentes que influenciem o desempenho do sistema em avaliação. Entre os possíveis critérios de avaliação, podem ser ressaltados: tempo de resposta, taxa de processamento e utilização de recursos. Esses critérios serão posteriormente transformados em métricas objetivas que possibilitam sua mensuração. O Modelo Abstrato é a base para geração de modelos que possuirão aspectos temporais; pois o modelo abstrato não possui nenhuma informação temporal associada.

Na fase de validação e análise qualitativa, os modelos (Abstrato e Refinado) são validados qualitativamente, ou seja, analisando-se as propriedades

e o comportamento dos componentes dos modelos (Abstrato e Refinado). Verificando se esses modelos atendem os requisitos iniciais do sistema avaliado, para que sejam feitos possíveis ajustes.

Na fase de medição, elabora-se, inicialmente, um “protocolo” de medição composto do planejamento, coleta e análise dos dados, como também eventuais recomendações. Na subfase de planejamento esse “protocolo” é definido. Na subfase da coleta dos dados, definem-se os atores responsáveis pela execução das atividades, onde, em que formato deve ser armazenado. Para a subfase de análise dos dados, geram-se estatísticas convenientes, as quais fornecem informações precisas a respeito do sistema em avaliação. Na última subfase, apresenta-se um conjunto de distribuições que melhor representam as atividades mensuradas.

Na definição das métricas de desempenho baseia-se nos critérios de desempenho definidos anteriormente. Não existe um conjunto padrão de métricas que devem ser utilizadas, uma vez que essa escolha depende da particularidade de cada sistema e do estudo que se pretende realizar. Ao final desta fase, serão definidas as métricas a serem consideradas na avaliação.

Na etapa de geração do modelo refinado, o Modelo Abstrato é transformado, com base nas estatísticas obtidas, considerando o tipo de distribuição sugerida na fase de medição. Ao final, serão gerados modelos refinados, os quais poderão ser avaliados e os resultados numéricos referentes a cada métrica analisados. Posteriormente, na fase de validação quantitativa, verificando-se se os números obtidos em relação às métricas representam a realidade.

Na fase de validação quantitativa, os modelos gerados na fase anterior serão validados através de comparação dos resultados obtidos através dos modelos com os providos por medições realizadas nas aplicações reais, a fim de que os modelos sejam uma representação do sistema em estudo.

Na fase de avaliação, o método (simulação/análise estacionária ou transiente) é definido e posteriormente a avaliação é realizada, considerando-se os experimentos, as métricas e os modelos definidos em fases anteriores. A fase

de análise dos resultados é a última etapa da metodologia e tem como objetivo apresentar os resultados obtidos na avaliação e interpreta os dados, ressaltando os pontos positivos e negativos do sistema, sugerindo eventuais ajustes ao sistema avaliado. A apresentação desses resultados e conclusões deve ser feita de forma direta e clara, de modo que as pessoas possam compreender os resultados numéricos obtidos.

5.2. MEDIÇÃO

O principal pré-requisito para realização do procedimento de medição é a definição de um documento que contém o planejamento da medição, ou seja, um documento descritivo de todo o procedimento para a realização da medição, contendo as fases de planejamento, coleta, análise e recomendações. A Figura 5.2 apresenta este procedimento através do diagrama de atividades [20].

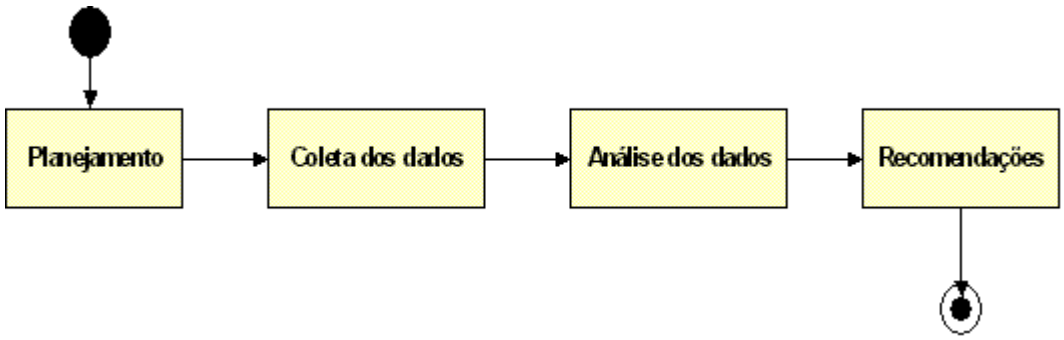


Figura 5-2 - Fluxo da Medição.

- Planejamento: tem como resultado um documento que descreve o protocolo de medição discriminado na Tabela 5.1. Esse documento deve conter informação sobre “o quê”, “onde”, “como”, “quem”, “quando” e a “freqüência” do processo de coleta dos dados, assim como a forma de armazená-los, o plano de análise e quem deverá realizar cada tarefa. Esse protocolo deve ser bastante conhecido por todos aqueles que deverão aplicá-lo.

Tabela 5.1 – Protocolo de Medição.

O quê	Onde	Como	Quem	Quando e Freqüência
Um conjunto de critérios subjetivos (do usuário, cliente) deve ser transformado em métricas.	O avaliador deve definir onde realizar as medições. Processos menos detalhados (ou imaturos) normalmente consideram a medição no final do processo, enquanto processos mais detalhados (ou mais maduros) consideram medição dentro do processo.	O avaliador especifica detalhadamente como realizar as medidas, o que inclui também o processo de calibração dos equipamentos e ferramentas (computacionais) a serem utilizadas.	A pessoa responsável pela execução desse protocolo de medição.	O quando e as freqüências das medições devem garantir a representatividade dos dados coletados.

- Coleta dos Dados: corresponde a definição do ambiente de medição, (a configuração da maquina utilizada e os softwares utilizados), como também, a calibração do das ferramentas computacionais. Nesta etapa define-se o formato que os dados serão armazenados para posterior análise.
- Análise dos Dados: corresponde à aplicação de métodos estatísticos para a análise dos dados coletados, apresentando-os de maneira adequada, ressaltando os resultados das estatísticas geradas, por exemplo, a média, o desvio padrão, a mediana, o coeficiente de variação e os quartis. Nessa fase, apresenta-se o método adotado em [61] para a geração de estatísticas considerando intervalos de confiança quando o desvio padrão é desconhecido (ver Figura 5.4). Para manter o intervalo de confiança com um nível desejado, compensa-se essa incerteza adicional fazendo o intervalo de confiança um pouco mais largo, usando os valores críticos fornecidos pela distribuição t de *Student*, desenvolvido por William Grosset [61].

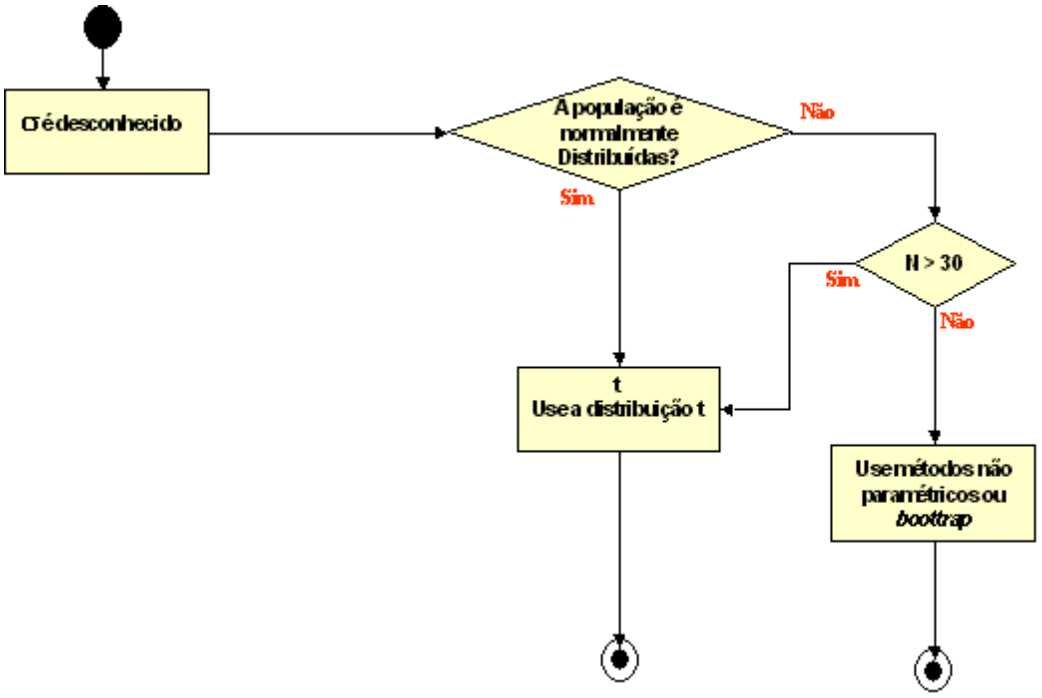


Figura 5-3 - Fluxo da definição do desvio padrão.

Ao aplicar testes estatísticos, antes de tudo, deve-se estabelecer o nível de significância (alfa), que depende do tamanho da amostra e da variabilidade interna das variáveis em estudo na amostra. Ao se determinar o nível de significância numa amostra, deve-se levar em consideração também os pontos fracos da pesquisa, como o controle de variáveis, confiabilidade e fidedignidade dos instrumentos de medição.

O procedimento de retirada dos *outliers* [18] é composto por seis etapas e requer um processo a ser seguido, pois seus valores revelam importantes informações e pode afetar grandemente os valores da média e do desvio padrão. A primeira etapa consiste em ordenar os valores e calcular as estatísticas como média, desvio padrão, variância, mediana, coeficiente de variação e quartis. A segunda etapa consiste em analisar as estatísticas, verificando os valores de centro e de dispersão. A quarta etapa consiste no cálculo de intervalo inter-quartis. A quinta etapa consiste na retirada dos *outliers* suspeitos com base nos valores calculados na etapa anterior. A última etapa consiste na geração das novas estatísticas sem *outliers* suspeitos.

- Recomendações: esta etapa ocorre depois da retirado dos *outliers* suspeitos e tem como objetivo utilizar *moment matching* [15], sugerindo um conjunto de distribuições (hipo-exponencial, hiper-exponencial, exponenciais, deterministicas ou Erlang) com base no conjunto de estatíticas (média e desvio padrão) obtidos na fase de medição. Portanto, modelos refinados são gerados para posterior avaliação.

5.3. GERAÇÃO DO MODELO ABSTRATO

Essa seção apresenta de maneira sucinta a fase de geração do modelo abstrato do sistema avaliado. A geração deste modelo permite a visualização, validação inicial e documentação do sistema avaliado. A escolha de qual modelo construir tem uma profunda influência na solução delineada. Todo modelo pode ser expresso em diferentes níveis de precisão. A escolha da granularidade determina as análises que podem ser realizadas, dependendo do detalhamento dos componentes dos modelos.

A geração do modelo abstrato é definida com base nos componentes e critérios definidos nas fases anteriores. O processo de modelagem é fundamental na geração dos modelos, possibilitando validação e análise qualitativa, de maneira que os modelos refinados possam ser construídos a partir de modelo “abstrato” com boas características, tais quais, ausência de *deadlock*, reversibilidade, limitação etc. Inicialmente, modela-se cada componente do sistema, levando-se em consideração o nível de granularidade adequado.

5.4. VALIDAÇÃO E ANÁLISE QUANTITATIVAS

Essa fase ocorre sempre que um modelo é gerado, ou seja, após as fases de geração do modelo abstrato e geração do modelo refinado. Após a validação poderá existir a necessidade de ajustes ou não nos modelos. Caso seja necessários ajustes, retorna-se à fase de geração do modelo (Abstrato e Refinado), para que modificações sejam realizadas e o modelo seja novamente validado e analisado qualitativamente. Caso não sejam necessários ajuste, o

modelo está qualitativamente validado. Tendo-se validado e analisado qualitativamente o Modelo Abstrato; e após execução da fase de medição, o Modelo Refinado é obtido, para ser validado e analisado qualitativamente.

5.5. GERAÇÃO DO MODELO REFINADO

Nessa seção, o modelo abstrato inicialmente proposto pode gerar vários modelos refinados em função das estatísticas obtidas na fase de medição. Estas estatísticas sugerem o tipo de distribuição associada, levando-se em consideração dois momentos: média e desvio padrão. Com os dados coletados e as estatísticas obtidas na fase de planejamento da medição, utiliza-se a técnica de *moment matching* [15] para aproximar distribuições empíricas através de distribuições poli-exponenciais (aproximação por fase, ver Seção 3.9). Desta forma, duas atividades são fundamentais, primeiro determina-se o tipo de aproximação necessária e, segundo, encontram-se os parâmetros numéricos da aproximação. Existem pontos considerados importantes na escolha de uma aproximação por fase, como a qualidade da aproximação, as medidas de aproximação, o número de estados da aproximação e a facilidade de obtenção do modelo *semi-markoviano* resultante.

5.6. VALIDAÇÃO QUANTITATIVA

Nessa etapa, os modelos refinados já validados qualitativamente, são validados quantitativamente, ou seja, os resultados obtidos na modelagem e os mensurados na aplicação são confrontados para verificar se o modelo refinado reflete as características da aplicação mensurada. Ao final desta etapa, os modelos estão validados e futuras avaliações poderão ser realizadas sem que haja perda de precisão dos resultados obtidos para métricas em questão.

Percebe-se, portanto, que os modelos propostos representam aplicação real e pode ser avaliado mesmo que aplicações ainda estejam em fase de projeto. Diversas métricas podem ser avaliadas e seus resultados interpretados, sugerindo recomendações para melhorar o desempenho do sistema em estudo.

5.7. AVALIAÇÃO E ANÁLISE DOS RESULTADOS

Nessa fase serão aplicadas as técnicas de avaliação de desempenho, levando em consideração os requisitos de hardware disponíveis, a estrutura dos modelos refinados e as métricas que se deseja avaliar. Essas técnicas podem ser: simulação/análise de estado estacionário ou transiente.

Uma avaliação transiente analisa o comportamento do modelo a partir da marcação inicial no tempo zero até um instante especificado. Conseqüentemente, um modelo pode ser usado para responder perguntas do tipo: qual é a probabilidade que após uma semana da operação, o sistema ainda esteja operando?; ou quantas mensagens são enviados durante uma hora para um servidor?

A avaliação de estado estacionário computa o desempenho médio do sistema depois que todos os efeitos transientes inicias passaram, e um estado de equilíbrio tenha sido alcançado, se for caso. Nessa avaliação a média das medidas de desempenho podem ser usadas para responder perguntas típicas, como: qual será o número previsto de clientes esperando para serem servidos. A seleção da avaliação transiente ou de estado estacionário deve ser feita inicialmente.

A análise é significativa uma avaliação de desempenho numérica direta e exata através de exploração do grafo de alcançabilidade (Cadeia Markov). Os algoritmos da simulação não computam o grafo de alcançabilidade (Cadeia de Markov), mas seguem a aproximação padrão da simulação do estilo Monte Carlos com alguns refinamentos [15]. Para definir entre qual técnica de avaliação utilizar, deve-se considerar a quantidade de recursos de hardware disponíveis, visto que, na análise, uma maior quantidade de memória pode ser necessária para o armazenamento do espaço de estados, porém seus resultados são exatos. Simulação, por outro lado, não demanda armazenamento substancial, contudo seus resultados são aproximações.

Na fase de análise dos resultados, inicialmente, justifica-se a escolha da técnica de avaliação utilizada, devido a dois aspectos fundamentes: a estrutura do modelo e os recursos disponíveis. Assim, os resultados obtidos após aplicação da

técnica de avaliação são representados graficamente para que recomendações sejam sugeridas para evitar, por exemplo, que o servidor ou *middleware* avaliados passem boa parte do tempo inativo.

5.8. CONSIDERAÇÕES FINAIS

Neste capítulo foi apresentada a metodologia para avaliação de desempenho de sistemas computacionais. Essa metodologia permite uma avaliação passo a passo e é iniciada a partir da compreensão geral do sistema avaliado, identificação de seus componentes e critérios de desempenho que serão utilizadas na geração do modelo abstrato. Para alimentar esse modelo, elaborou–se um documento, na fase de medição chamado Protocolo de Medição, que especifica cada etapa dessa fase. A partir do modelo abstrato, deriva-se um modelo refinado que inclui os aspectos temporais através de estruturas especificas. Desta forma, a metodologia proposta apresenta o conjunto de atividades definidas para o processo de avaliação.

6. MODELOS PROPOSTOS

Este capítulo apresenta os modelos para o Serviço de Controle Concorrência (SCC) do CORBA. Inicialmente, apresentam-se os componentes do SCC e suas operações. Posteriormente, um modelo abstrato deste serviço é proposto e discutido. Em seguida apresenta a aplicação do processo de medição, a geração dos modelos refinados e respectiva validação.

6.1. DEFINIÇÃO DO SISTEMA E SEUS COMPONENTES

Esta seção contempla a primeira etapa da metodologia definida no Capítulo 5 e consiste na apresentação do sistema a ser avaliado, levando em consideração seus limites e as suas características.

No CORBA, a comunicação entre cliente e servidor é usualmente síncrona, ou seja, o cliente faz uma solicitação e espera uma resposta. Essa solicitação é encaminhada para o *middleware*, que obtém a resposta no servidor e depois devolve ao cliente. Objetos cliente comunicam-se com objetos servidores através do *Object Request Broker* (ORB). Quando o cliente invoca um método num objeto remoto, o ORB localiza a instância do objeto servidor, invoca o método e retorna o resultado ao objeto cliente.

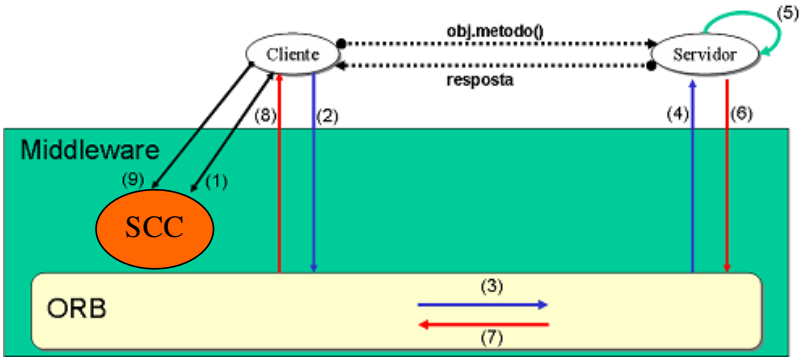


Figura 6-1 - Identificação dos Componentes.

1. O cliente solicita um *lock* (SCC)
2. O cliente invoca um método remoto (via ORB)
3. O ORB transporta a invocação para o servidor

4. O servidor recebe a invocação
5. O servidor processa a invocação
6. O servidor envia o resultado para o cliente (via ORB)
7. O ORB transporta o resultado da invocação para o Cliente
8. O SCC libera o *lock*
9. O Cliente recebe o resultado

Para se iniciar a modelagem do sistema, distinguem-se os principais componentes do serviço a ser modelado, conforme passos apresentados na Figura 6.1, para isso identificam-se:

- Cliente – tem como papel fazer solicitação para iniciar a leitura.
- *Middleware* – é uma infra-estrutura que visa à interoperabilidade de aplicações de forma transparente.
- SCC – é um serviço oferecido pelo CORBA, responsável pelas operações do *lock* (criação) e *unlock* (liberação).
- Servidor – é responsável pela execução das solicitações do cliente.

6.2. MODELO ABSTRATO

Esta seção apresenta o modelo inicial para o SCC baseado no formalismo das Redes de Petri [2]. Assim, o cliente, o *middleware* e o servidor apresentado na Figura 6.1 são modelados e apresentados na Figura 6.2.

O cliente é composto por vários componentes, são eles: um lugar (Cliente) e uma transição (Solicitação) que representa a realização das solicitações; duas transições imediatas (Escrita e Leitura) que definem o modo a ser utilizado (leitura ou escrita), possuindo uma probabilidade que varia de 0% a 100%, além de um outro lugar (Numero_Clientes_Lendo) com uma marcação (Numero_Leituras_Consecultivas) que representa o número de leituras consecutivas que o cliente realizará com apenas um *lock*.

O modelo do SCC do CORBA apresentado na Figura 6.2 é formado por duas transições gerais (Novo_Lock e *Unlock*) que ainda não foram definidas, pois dependem das estatísticas obtidas na fase de medição. Essas duas transições

representam o tempo de criação e liberação do *lock*, além de um lugar (*middleware*) representado um buffer, indicando quantos clientes podem ser atendidos simultaneamente. As transições imediatas (T1 e T2) possuem probabilidade associadas que dependem da marcação inicial e do número de clientes tem um *lock*. O lugar (Tem_lock) representa a detenção do *lock*. As transições imediatas (T5 e T6) possui probabilidades associadas, variando de 0% a 100%, e indicam a possilibidade de continuar com o *lock* e liberar o *lock*, respectivamente. A transição exponencial (Write_Lock) representa um resumo das operações de escrita (solicitação de escrita,criação do *lock*, tempo de escrita e liberação do *lock*) e o peso do arco de entrada e saída do lugar (Exclusão_Mutua) está relacionado a marcação do lugar Solicita_Leitura, indicando que quando um cliente está escrevendo nenhuma leitura poderá acontecer.

O modelo do servidor da Figura 6.2 é formado basicamente por uma transição exponencial (TX_Servidor), que indica o tempo necessário para realização de uma operação de leitura ou escrita e um lugar (Servidor) representando um buffer, indicando quantos clientes podem ser servidos simultaneamente. O lugar (Inicio_Servidor) representa o número de clientes esperando para serem servidos.

Um *token* no lugar Cliente indica que uma solcitação está sendo feita, e poderá ser de escrita ou de leitura depedendo das probalidades associada às transições (Escrita e Leitura). Se for de leitura aparecerá um *token* no lugar Chega_Solitação, caso seja de escrita, aparecerá no lugar Modo_Write. Caso seja necessária a criação de um lock, habilitará a transição Novo_Lock e um novo lock será criado, habilitando a transição T3. Caso não haja a criação de um lock, a transição T2 é habilitada e um *token* colocado no lugar Tem_Lock e transição T8 é habilitada. Assim, uma leitura ocorrerá e transição TX_Servidor estará habilitado. Caso uma solicitação de escrita seja solicitada, habilitando a transição T12, indicando que apenas essa operação será realizada pelo servidor, os demais clientes que estiverem lendo ou esperando para ler serão retirados do servidor.

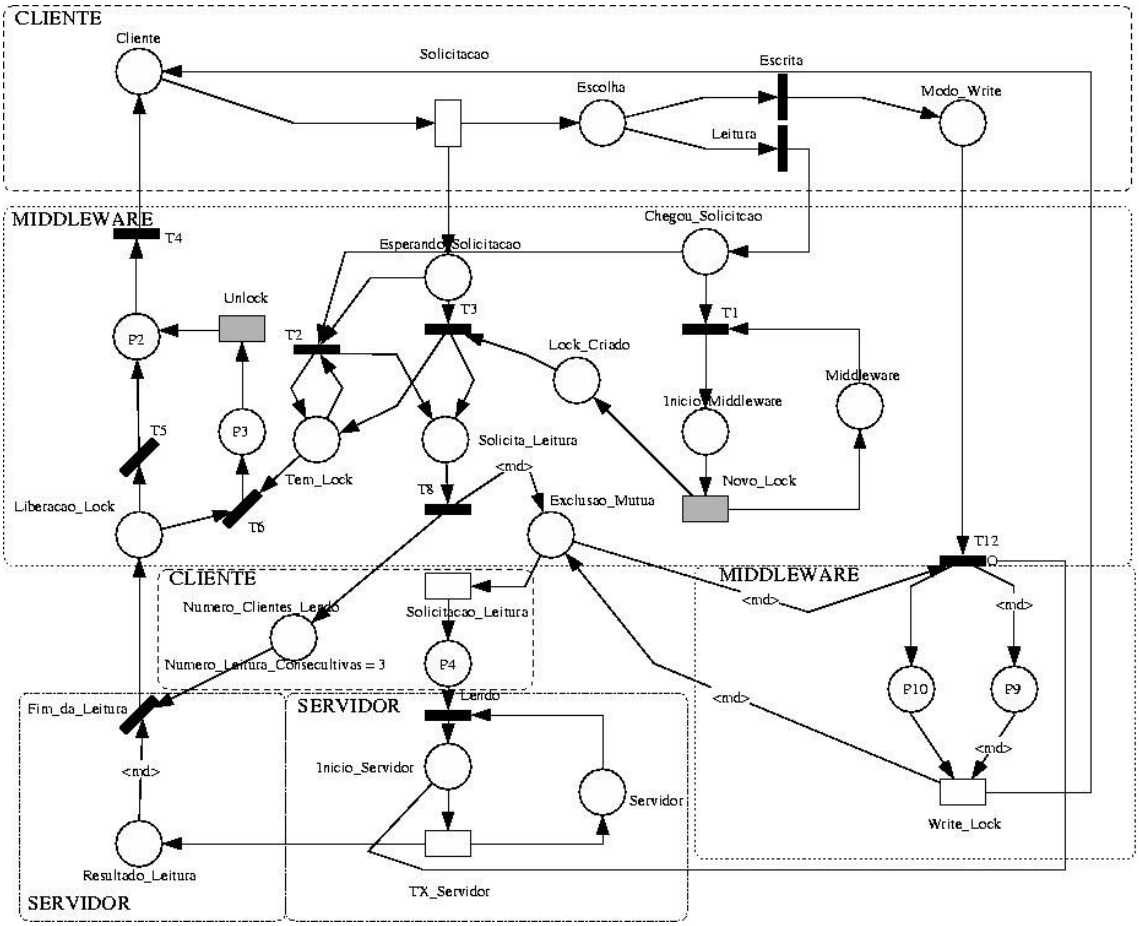


Figura 6-2 - Modelo Abstrato.

6.3. MEDIÇÃO

Essa seção descreve o protocolo de medição definido no Capítulo 5. Esse protocolo será definido na fase de planejamento da medição e possuirá um conjunto de atividades ordenadas (planejamento, coleta, análise e recomendações). Cada etapa da medição será planejada antes de ser executada e, através desse planejamento, poder-se-á escolher, por intermédio das estatísticas obtidas, qual a distribuição mais adequada para a amostra em questão. As recomendações serão sugeridas em relação ao modelo apresentado na Figura 6.2; e as transições gerais, para as operações de criação e liberação do *lock* serão substituídas através das especificações temporais incorporadas ao modelo. Portanto, modelos refinados serão gerados a partir do modelo proposto inicialmente.

6.3.1. Planejamento

O planejamento da medição segue o conjunto de recomendações apresentada no Capítulo 5, de forma que as questões mencionadas na Tabela 5.3 sejam respondidas (O que medir? Onde medir? Quem vai medir? Como medir? Quando e Qual a freqüência da medição?). Primeiramente, definiu-se o profissional que deve realizar a medição. Esta etapa foi feita pelo próprio planejador. Posteriormente, definiram-se as métricas básicas a serem medidas. As métricas básicas consideradas foram: o tempo de criação e liberação das operações *lock* e *unlock* dos modos *read* e *write*. A unidade de tempo adotada doi o milissegundo. Foram realizadas 12.000 medições consecultivas. Para obter essa amostra (12.000 dados) foi utilizada a biblioteca *HRTimer* implementada pela interface ITimer do Java através de quatro métodos. O método *start*() inicia novo intervalo de tempo e o método *stop*() finaliza esse intervalo. O método *getduration*() retorna a duração do intervalo de tempo decorrido entre a chama da dos métodos *star*() e *stop*(). Finalizando, o método *reset*() é utilizado para resetar o contador, permanecendo em estado de pronto. Os experimentos foram todos realizados em uma mesma máquina, ou seja, o servidor de concorrência do CORBA/OpenORB[8], o servidor de nomes e as aplicações clientes estavam executando na mesma máquina, com a seguinte configuração:

- Pentium III de 800 MHz, com HD de 20 GB (5400 RPM Ultra-ATA/100), e 190 MB de memória RAM. O computador utilizava o sistema operacional Windows 2000 Professional (versão 5.00.2195 – Service Pack 4);
- Ambiente de desenvolvimento, Java 2 SDK, *Standard Edition*, versão 1.4.2_02;
- A versão 1.4.0 BETA2 *Released* do OpenORB[8].

Com isso, calcularam-se as principais medidas de tendência central (média, mediana e moda) e as medidas de dispersão (coeficiente de variação, desvio padrão e variância). Entretanto, para esse trabalho considerou o primeiro e o segundo momento para obtenção dos modelos refinados, levando em

consideração para geração desses modelos às estatísticas (a média e o desvio padrão).

6.3.2. Coleta dos dados

Após o planejamento dos dados, a medição das métricas básicas (média, desvio padrão) foi realizada e as amostras armazenadas em arquivos compatíveis com o Microsoft Excel. Para que se obtivesse a menor interferência no momento da coleta dos dados, encerram-se todos os processos que não eram estritamente necessários para correta execução das atividades operacionais do sistema. A Figura 6.4 apresenta o conjunto de processos ativos que precedeu o momento da coleta dos dados. Percebe-se que o sistema está 99% desocupado ,anteriormente, a execução da medição, garantindo que o menor quantidade de processos estavam ativos no momento da medição.

O tamanho da amostra utilizada foi de 12.000 dados. No entanto, as 2.000 primeiras amostras foram ignorados, pois observou-se que os primeiros dados coletados apresentavam um comportamento tendencioso, possivelmente, devido a ao comportamento transitório da máquina virtual do Java [34]. Assim, esses dados foram descartados e a amostra analisada foi de 10.000 dados.

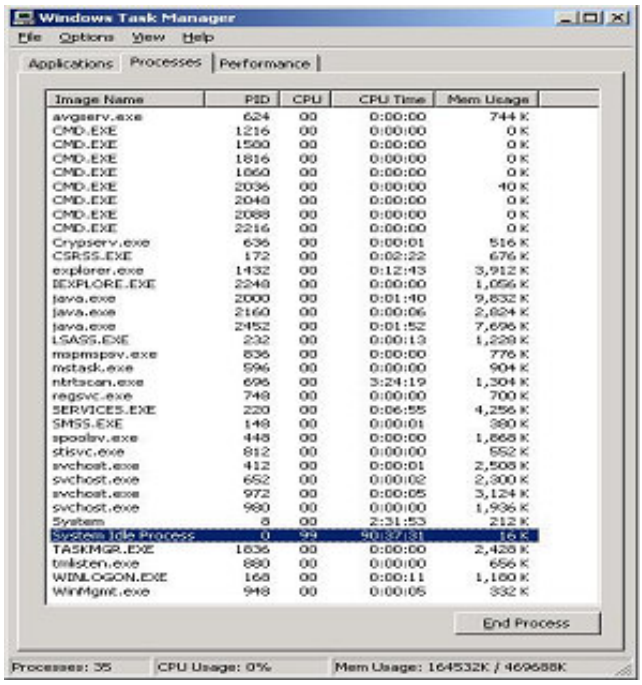


Figura 6-3 - Processos Rodado no momento da Coleta de Dados.

6.3.3. Análise dos dados

Conforme a metodologia adotada, nesta etapa utiliza-se a amostra obtida na etapa da coleta dos dados para calcular o tempo médio de criação da operação *lock* (Tabela 6.1) e *unlock* (Tabela 6.3) no modo de leitura, conforme estatísticas obtidas. Analisando-se os valores das medidas de dispersão e de tendência central, percebe-se que existe uma variabilidade dos valores obtidos, conforme a Tabela 6.1 para o cálculo da operação de *lock*. Pois, o coeficiente de variação obtido está em torno de 2, o valor máximo é 250 vezes maior que o valor mínimo, a amplitude é alta também e está em torno de 2,34, a média calculada está em torno de 0,012 e a mediana calculada é aproximadamente 0,011. A moda calculada é aproximadamente de 0,012. Portanto, observa-se que na amostra utilizada, existem alguns valores extremos chamados de *outliers*.

Tabela 6.1 - Operação *lock* no modo de leitura

Estatítica	Valor Absoluto		Estatítica	Valor Absoluto
Média	0,012080		**Máximo**	2,350019
Desvio Padrão	0,024392		**Mínimo**	0,008381

Amplitude	2,341638		**Moda**	0,012013
Coef. de Variação	2,019192		**Quartil(Primeiro)**	0,011174
Variância	0,000595		**Quartil(Terceiro)**	0,012292
Mediana	0,011733			

Para a retirada desses *outliers*, calcula-se o intervalo inter-quartil (IIQ) e considerou-se *outlier*, valores acima de 16 – 1,5*IIQ e abaixo de 3Q + 1,5*IIQ. Novas estatísticas foram calculadas e são apresentadas na Tabela 6.2. Observa-se que os novos valores centrais (média e mediana) obtidos são aproximadamente iguais e estão em torno de 0,011; o coeficiente de variação decresce consideravelmente, ficando em torno de 0,069 e a variância é 0,000001, indicando uma menor variabilidade. O valor máximo é agora 1,4 vezes maior que o valor mínimo. Assim, a nova amostra (sem os *outilers*) possui 9340 dados.

Tabela 6.2 - Operação *lock* sem *outliers* no modo de leitura.

Estatística	Valor Absoluto		Estatística	Valor Absoluto
Média	0,011676		**Máximo**	0,013968
Desvio Padrão	0,000806		**Mínimo**	0,009498
Amplitude	0,004470		**Moda**	0,012013
Coef. de Variação	0,069020		**Quartil(Primeiro)**	0,011175
Variância	0,000001		**Quartil(Terceiro)**	0,012292
Mediana	0,011733			

A Tabela 6.3 apresenta as estatísticas associadas ao tempo da operação *unlock* no modo de leitura. A média e o desvio padrão serão utilizadas para definir o tipo de distribuição que será utilizada no modelo refinado na Seção 6.6.

Diante das estatísticas calculadas, percebe-se que existe alta variabilidade nos dados obtidos, pois o coeficiente de variação está em torno de 4. O valor máximo é 500 vezes maior que o valor mínimo e a amplitude é alta e está em torno de 3,63. A mediana é aproximadamente igual a 0,010 e a média é próxima a 0,011 e a moda é aproximadamente 0,010.

Tabela 6.3 - Operação *unlock* no modo de leitura.

Estatística	Valor Absoluto		Estatística	Valor Absoluto

Média	0,011185		**Máximo**	3,643200
Desvio Padrão	0,045813		**Mínimo**	0,007263
Amplitude	3,635937		**Moda**	0,010057
Coef. de Variação	4,095860		**Quartil(Primeiro)**	0,009499
Variância	0,002099		**Quartil(Terceiro)**	0,010616
Mediana	0,010057			

Como no caso anterior, observa-se a presença de *outliers* suspeitos, desta forma, estes valores foram removidos da amostra, seguindo procedimento anteriormente adotado. As novas estatísticas são apresentadas na Tabela 6.4. A média e a mediana calculadas nesta nova base de dados aproximam-se e ficam em torno de 0,010. O coeficiente de variação decresceu consideravelmente, ficando em torno de 0,075714. Desta forma, a variância ficou com 0,000001 e o valor máximo é 1,4 vezes maior que o valor mínimo. Estes valores extremos (retirados da amostra) estão provavelmente relacionados a intervenções da maquina virtual.

Tabela 6.4 - Operação unlock sem outliers no modo de leitura.

Estatística	Valor Absoluto		Estatística	Valor Absoluto
Média	0,010131		**Máximo**	0,010057
Desvio Padrão	0,000767		**Mínimo**	0,012013
Amplitude	0,003912		**Moda**	0,008101
Coef. de Variação	0,075714		**Quartil(Primeiro)**	0,009499
Variância	0,000001		**Quartil(Terceiro)**	0,010616
Mediana	0,010057			

6.3.4. Recomendações

Esta etapa recomenda um conjunto de alternativas do refinamento de acordo com as estatísticas obtidas na etapa anterior. Utiliza-se a técnica de *moment matching* [15] para definir distribuições adequadas para refinar o Modelo Abstrato em função da média e do desvio padrão. Adotando essa técnica, as distribuições recomendadas são hipoexponencial, com número de fases extremamente elevados(ver Tabela 6.5). O elevado número de fases recomendado dificulta a resolução numérica (solução de Cadeia de Markov) dado

o espaço de estado gerado e as restrições de memória das plataformas de avaliação.

6.4. DEFINIÇÃO DAS MÉTRICAS

As métricas adotadas para avaliar o desempenho do SCC do CORBA são:

- Vazão do *middleware*: quantas operações de *locks* podem ser criadas no modo de leitura por unidade de tempo;
- Vazão do sistema: quantas operações de leitura o servidor pode realizar por unidade de tempo;
- Capacidade do servidor: número de clientes sendo servidos simultaneamente;
- O tempo médio entre criação de *locks*, no modo de leitura.

Para obtenção dos valores dessas métricas, seis cenários foram propostos, visto que possui uma variedade de parâmetros que serão variados.

6.5. MODELOS REFINADOS

As estatísticas obtidas na Seção 6.3.3 (Análise dos Dados) foram utilizadas para definição dos tempos associados às transições Novo_Lock e *Unlock* dos modelos refinados. Conforme as alternativas apresentadas, estas transições genéricas foram substituídas por uma transição exponencial, uma sub-rede Erlang ou uma transição determinística.

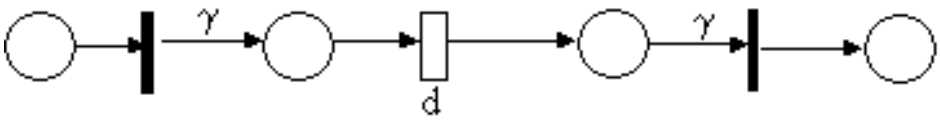


Figura 6-4 - Sub-rede Erlang.

A sub-rede Erlang considerada, que representa uma distribuição de probabilidade de Erlang, é composta por duas transições imediatas e uma exponencial (ver Figura 6.4) e as características temporais dessa sub-rede são apresentadas na Tabela 6.6.

Tabela 6.5 - Fases do Novo_Lock e Unlock.

LOCK_READ		UNLOCK_READ	
Valor		**Valor**	
Média	0,01174693	**Média**	0,00951843
Tempo da Transição Exponencial(d)	0,01174693/ γ	**Tempo da Transição Exponencial(d)**	0,00951843/ γ
Peso dos arcos (γ)	**257**	**Peso dos arcos (γ)**	**152**
Número de Fases	**257**	**Número de Fases**	**152**

Para obter o número de fases da distribuição, apresentado na Tabela 6.5, um conjunto de passos foi definido. Divide-se a média pelo desvio padrão, eleva-se este quociente ao quadrado para obter o número de fases da sub-rede (ver Seção 3.9). O valor obtido será utilizado como peso dos arcos (γ) de saída da primeira transição imediata e de entrada para a segunda transição imediata, o tempo da transição exponencial (d) é a média dividida pelo peso dos arcos (γ). Portanto, a transição genérica será substituída por uma distribuição Erlang com 257 fases para a transição Novo_Lock e 157 fases para transição *Unlock*. Esse número de fases dificulta a realização de experimentos devido ao número de estados gerados. No entanto, três alternativas foram apresentadas (ver Seção 6.4.3) para contornar a geração do espaço de estados recomendado.

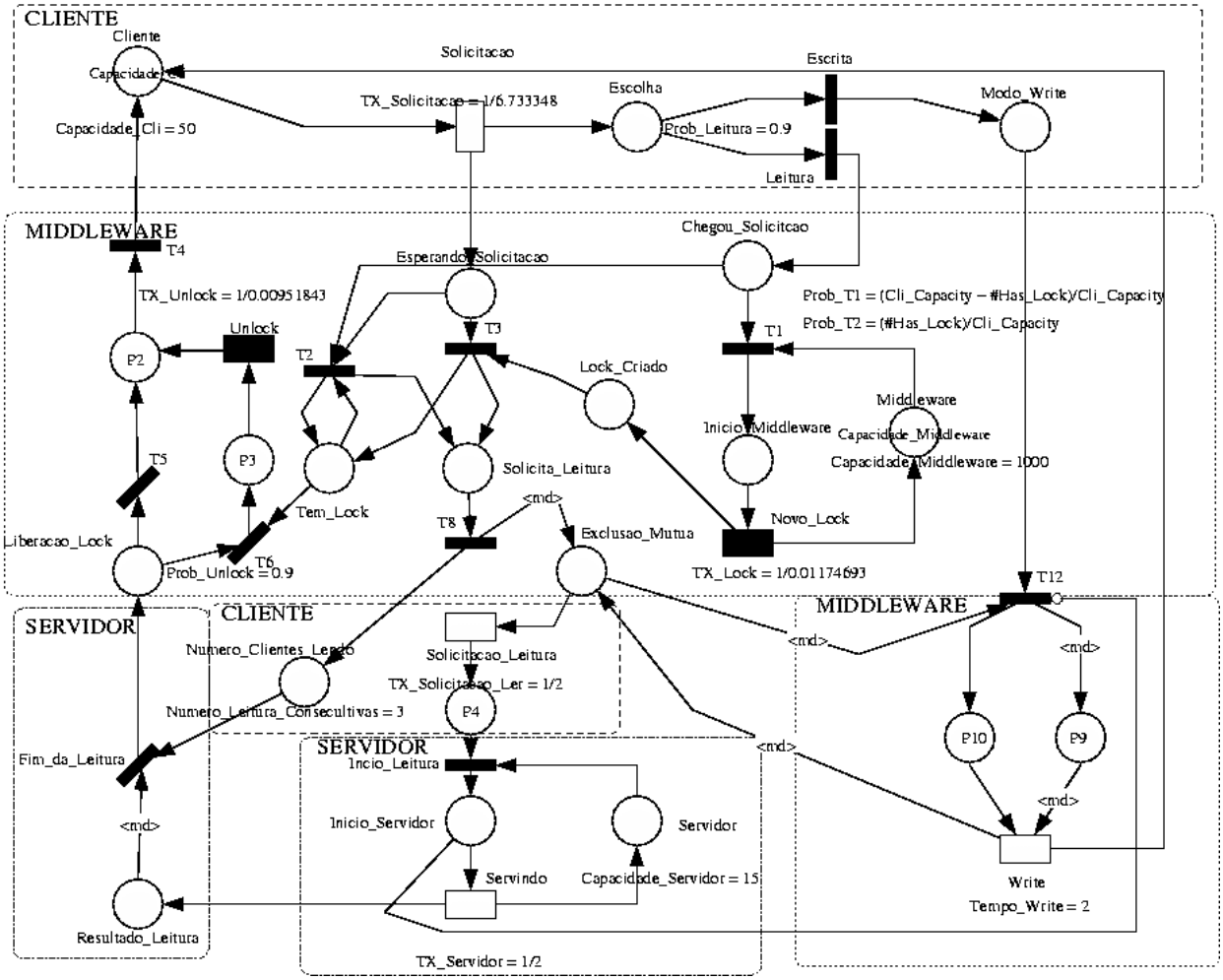


Figura 6-5 - Modelo com Transição Determinística.

Na primeira alternativa mostrada na Figura 6.5, utiliza transições determinísticas, para representar o tempo (média) das transições Novo_Lock e Unlock. Assim, a transição Novo_Lock, que inicialmente é representada por uma distribuição genérica, é substituída por uma transição determinística com tempo igual a 0,01174693 ms. A transição *Unlock* é substituída por uma transição determinística com tempo igual a 0,0091543 ms.

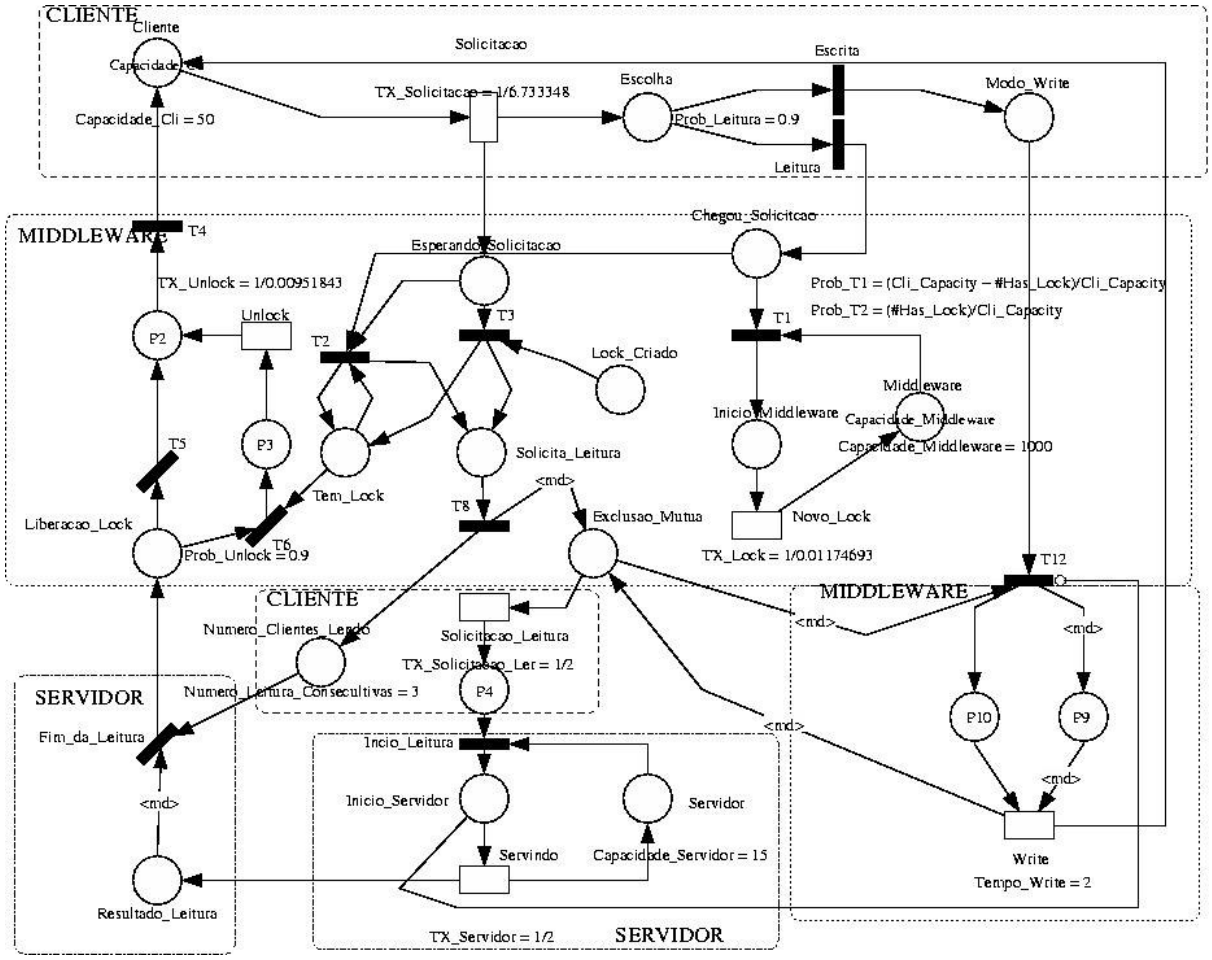


Figura 6-6 - Modelo com Transição Exponencial.

A segunda alternativa é a utilização da transição exponencial no lugar das transições Novo_Lock e *Unlock* (ver Figura 6.6). A utilização da distribuição exponencial representa um menor custo de avaliação quando utiliza-se a técnica de análise numérica. Assim, as transições Novo_Lock e *Unlock* são substituídas por transições exponenciais com tempo médio igual a 0,011676 e 0,01031, respectivamente.

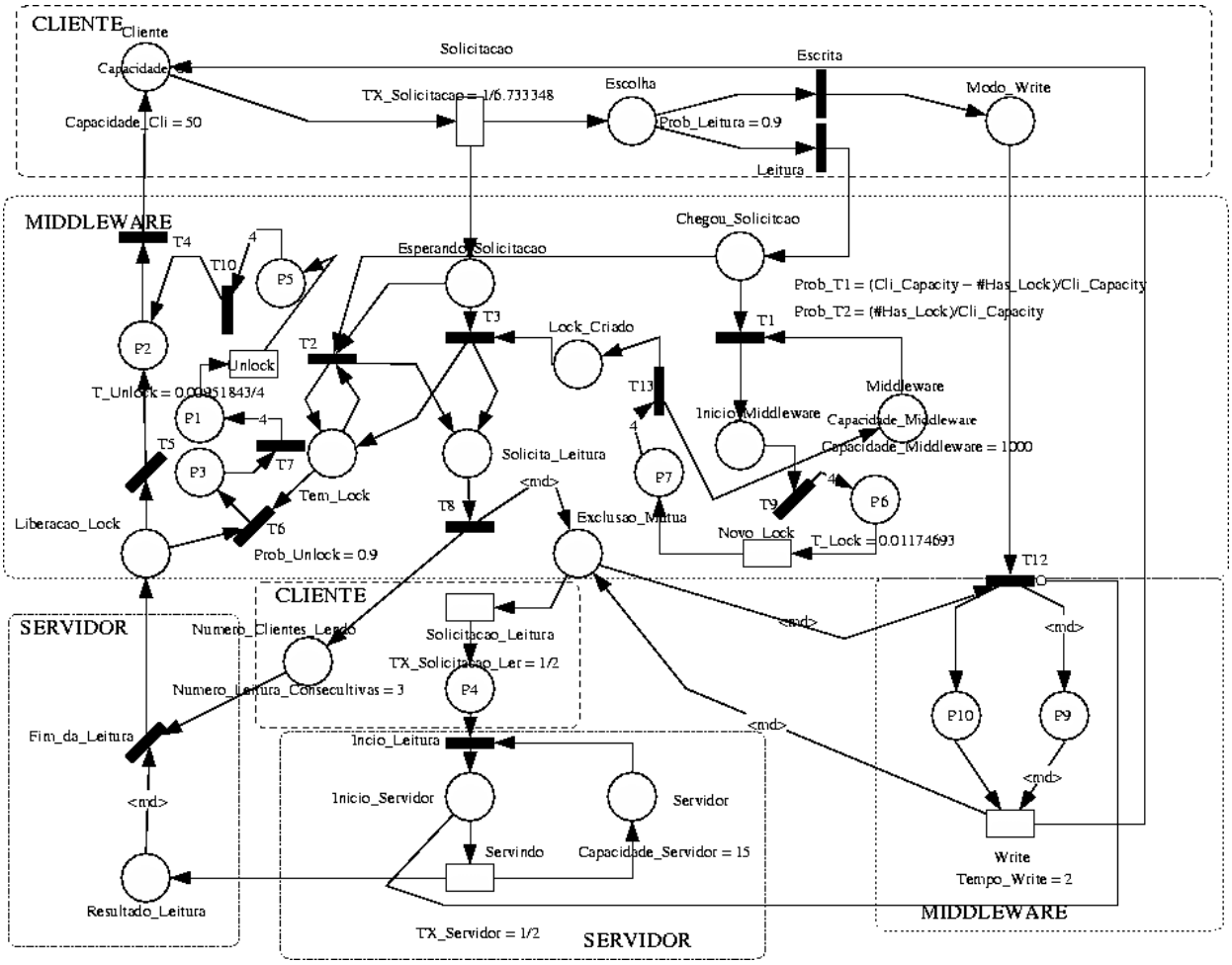


Figura 6-7 - Modelo Erlang.

A terceira alternativa mostrada na Figura 6.7 utiliza um número menor de fases que o recomendado no inicio desta seção. Foram realizados experimentos com sub-redes Erlang com 4, 6, e 8 fases. Nestes casos, a transição Novo_Lock foi substituída por uma sub-rede formada por duas transições imediatas T13 e T9, dois lugares P6 e P7 e uma transição exponencial com tempo igual a 0,0011676 / γ ms, onde g pode ser 4, 6 ou 8. Os pesos dos arcos (γ) de saída da transição T9 e de entrada da transição T13 poderão ser 4, 6 ou 8. A transição *Unlock* foi substituída por uma sub-rede formada por duas transições imediatas T7 e T10, dois lugares P3 e P1 e uma transição exponencial tempo igual a 0,01031/γ, onde "γ" pode ser 4, 6 ou 8. Os pesos dos arcos (γ) de saída da transição T7 e de entrada da transição T10 poderão ser 4, 6 ou 8.

6.6. VALIDAÇÃO

A validação é realizada para certificação da modelagem, ou seja, nessa etapa avalia-se o modelo obtido (que representa o SCC do CORBA). Para validar o modelo são confrontados dados obtidos pelo modelo com os dados mensurados. Para facilitar o processo de validação, um modelo simplificado, com as mesmas características dos modelos recomendados é adotado. A métrica adotada para validação é o tempo entre criação de *locks*. Nesta etapa validam-se o modelo determinístico e o baseado no sub-rede *Erlang*. Inicialmente, apresentam-se os resultados da validação do modelo determinístico. Depois, os resultados do processo de validação do modelo *Erlang* são apresentados.

O modelo proposto para realização dessa validação é apresentado na Figura 6.8, e foi desenvolvido para atuar em um cenário específico e simplificado. Para o cenário desenvolvido, calculou-se o tempo médio entre *locks*, variando-se a quantidade de clientes. Neste cenário, o cliente realiza solicitações a uma taxa predefinida de 1/6.7333 ms^{-1}, a probabilidade do modo de leitura é definida como 1, ou seja, só serão consideradas leituras. A taxa de criação do *lock* é 1/0,0117469 ms^{-1}, a taxa do servidor é 0,5 ms^{-1}. Considera-se que, apos a realização da leitura (disparos da transição Servindo), os *locks* são sempre liberados.

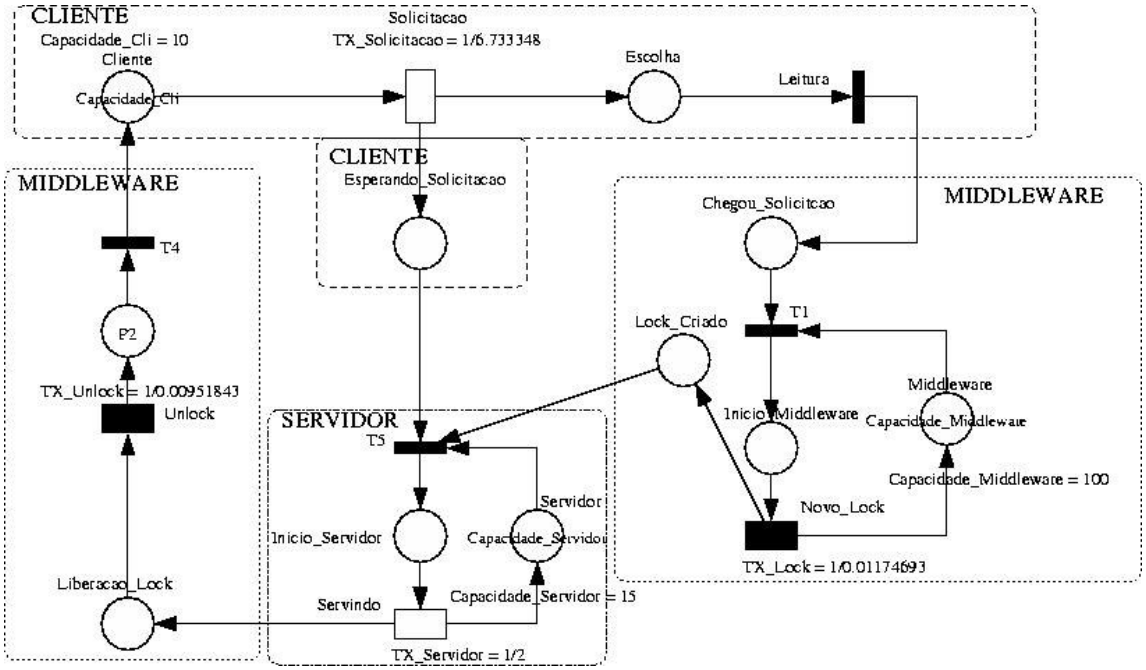


Figura 6-8 - Modelo Simplificado no Timenet.

A Figura 6.9a apresenta o tempo médio entre *locks* obtido na ferramenta TimeNET. Este valor situa-se em torno de 6.8 milessegundo. A Figura 6.9b representa o tempo médio entre *locks* medido no SCC do CORBA. Este valor situa-se entre os limites 7 e 5,2 ms, considerando até dez clientes. Com estas medidas verifica-se que o modelo com transições determinísticas apresentam informações de desempenho próximas dos modelos realizados no SCC do CORBA.

Observa-se que o tempo médio entre locks diminui à medida que aumentamos a quantidade de clientes, chegando a um determinado ponto que tende a estabilizar. Na Figura 6.9a, este valor estabiliza em 6,8 ms e, na Figura 6.9b, o valor obtido situa-se em torno de 5,2 ms, quando o numero de clientes superior a oito. A diferença percentual entre a aplicação CORBA e o modelo determinístico é de 24%(vinte e quatro por cento)

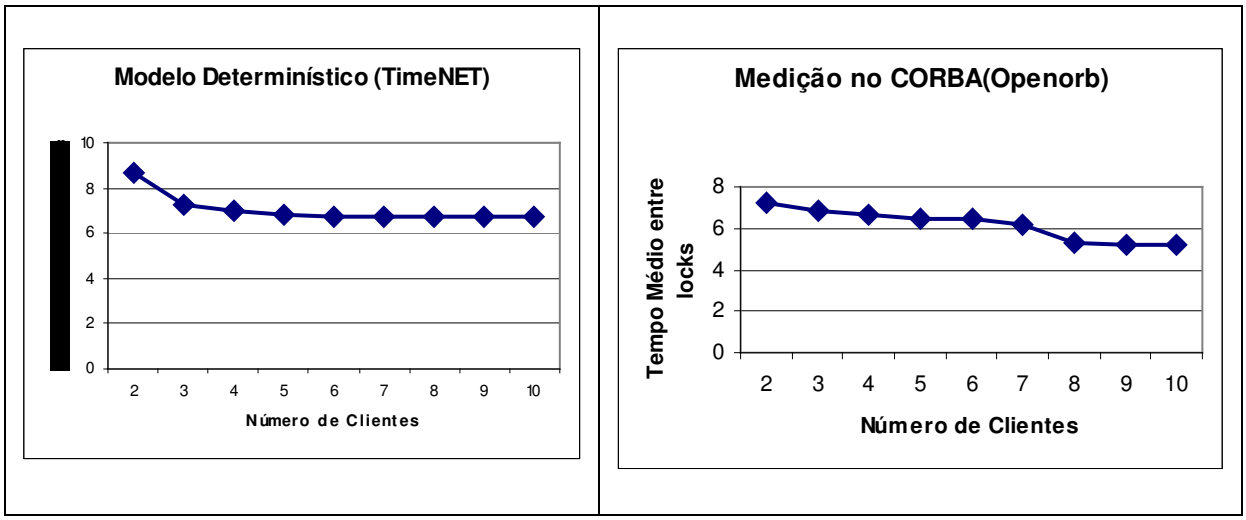


Figura 6-9a Tempo médio entre locks (Timenet) Figura 6.9b - Tempo médio entre locks(CORBA)

A Figura 6.10 apresenta os resultados (tempo entre *locks*) para o modelo baseado em sub-rede Erlang. Os resultados apresentados concernem a sub-rede Erlang com 4, 6 e 8 fases. As três configurações (4,6 e 8 fases), para este exemplo, fornecem resultados muito próximos.

Observou-se que, quando se aumenta o número de clientes, diminui-se o tempo médio entre solicitações de *locks*, estabilizado, ao chegar a 8 clientes, em torno de 6,8 milessegundos. Percebe-se claramente que se pode utilizar tanto uma sub-rede Erlang como a transição determinística para realização dos experimentos, pois os valores calculados são próximos e a métrica avaliada segue a mesma tendência em ambos os modelos utilizados.

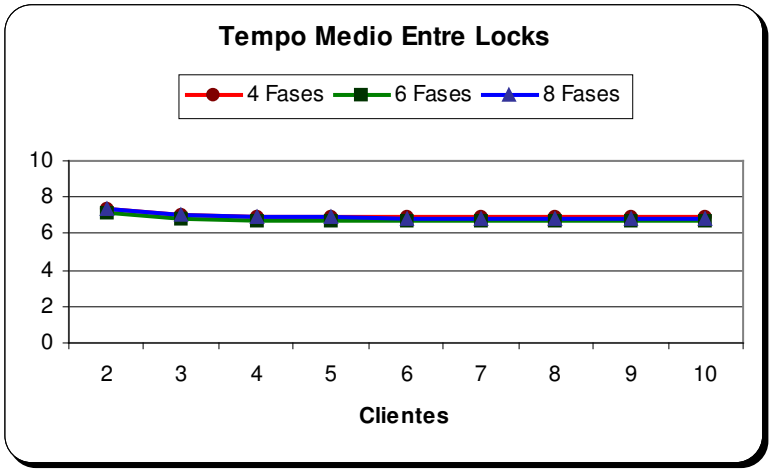


Figura 6-10 - Tempo Médio entre Locks usando Erlang.

6.7. CONSIDERAÇÕES FINAIS

Neste capítulo apresentou a aplicação da metodologia descrita no Capitulo 5 para modelagem, e validação do SCC do CORBA. A partir da especificação do sistema e seus componentes, gerou-se um Modelo Abstrato. Posteriormente, definiu-se um ambiente da medição do sistema a ser avaliado, para que o modelo abstrato seja refinado e modelos mais precisos sejam gerados. Com as estatísticas obtidas através da média e do desvio padrão, distribuições de probabilidade foram sugeridas. A primeira sugestão seria uma sub-rede *Erlang* com 257 fases para operação de *lock* e 152 fases para operação de *unlock*, esses valores dificultaram a realização dos experimentos devido ao número de espaço de estado gerados. Assim, três alternativas foram apresentadas. A primeira utiliza transição determinística e considerou apenas o valor da média como parâmetro para representação do tempo das operações de *lock* e *unlock*. Porém, utilizando a técnica de análise transiente, a primeira alternativa torna-se inviável, pois duas transições determinísticas não poderiam estar habilitadas em uma única marcação. Por isso, uma segunda alternativa foi utilizada, utilizou-se transição exponencial para representação das operações de *lock* e *unlock*. Essa alternativa possuía um custo menor e o tempo considerou as estatísticas da média e do desvio padrão. A terceira alternativa utilizada considerou um sub-rede Erlang com um número menor de fases do que o sugerido, utilizou-se 4, 6 e 8 fases. *lock* e

unlock. Portanto, ao final desse capítulo os modelos sugeridos como alternativas foram validados e seus resultados foram comparados, verificando-se tendências similares entre os mesmos.

7. ANÁLISE DOS EXPERIMENTOS

Neste capítulo, justifica-se a escolha da técnica de avaliação utilizada para obtenção das métricas. Posteriormente, apresenta-se um conjunto de cenários para realização das avaliações de acordo com a técnica escolhida. Posteriormente, os resultados são interpretados e recomendações ao sugeridas.

7.1. INTRODUÇÃO

Esta seção justifica a escolha da técnica de avaliação adotada. Para verificar qual técnica (simulação e análise estacionárias) a ser utilizada, um parâmetro (número de clientes) foi variado de maneira a verificar a aplicabilidade da técnica. Vale ressaltar que a aplicação de técnica baseada em análise estacionária pode se tornar inadequada dado a estrutura do modelo e a memória da máquina utilizada. Contudo, o número de clientes (parâmetro variável) pode ser uma restrição para aplicação dessa técnica, dada a memória disponível da máquina utilizada.

A configuração da máquina utilizada para avaliação foi um computador com processador *Athlon* XP 2.0 MHz, com HD de 40 GB e 528 MB de memória RAM. Com o sistema operacional *Linux*, distribuição *Gnome*, versão 2.6. A ferramenta *TimeNET*, versão 3.0 [4] e o Microsoft Excel do pacote do Office 2000. Observou-se que, a partir de sete clientes, a memória da máquina era insuficiente para o armazenamento da matriz geradora infinitesimal. Portanto, a simulação estacionária foi técnica adotada neste trabalho. Na utilização desta técnica considerou um intervalo de confiança de 95%, o número mínimo de disparos para cada transição de 50 e o erro relativo de 10%.

Nas subseções seguintes serão definidos os cenários para análise dos experimentos. Os modelos utilizados foram definidos no Capítulo 6. Para cada cenário, adotou-se os parâmetros fixos e variáveis. Por fim, os resultados são interpretados e possíveis recomendações são sugeridas.

7.2. PRIMEIRO EXPERIMENTO

Este experimento consiste na avaliação da vazão do sistema e do *middleware* em função do número de clientes. O Modelo Abstrato, apresentado na Figura 7.1, foi utilizado nesse experimento com medidas temporais associadas às transições utilizadas. Assim, a transição de distribuição genérica Novo_Lock foi substituída pela transição exponencial (ver Figura 6.6) e os resultados da análise estacionária são apresentados nas Figuras 7.2 e 7.4. Os resultados apresentados nas Figuras 7.3 e 7.5 foram obtidos pela análise do modelo, substituindo-se as transições genéricas Novo_Lock e *Unlock* por sub-redes com distribuição Erlang (ver Figura 6.7).

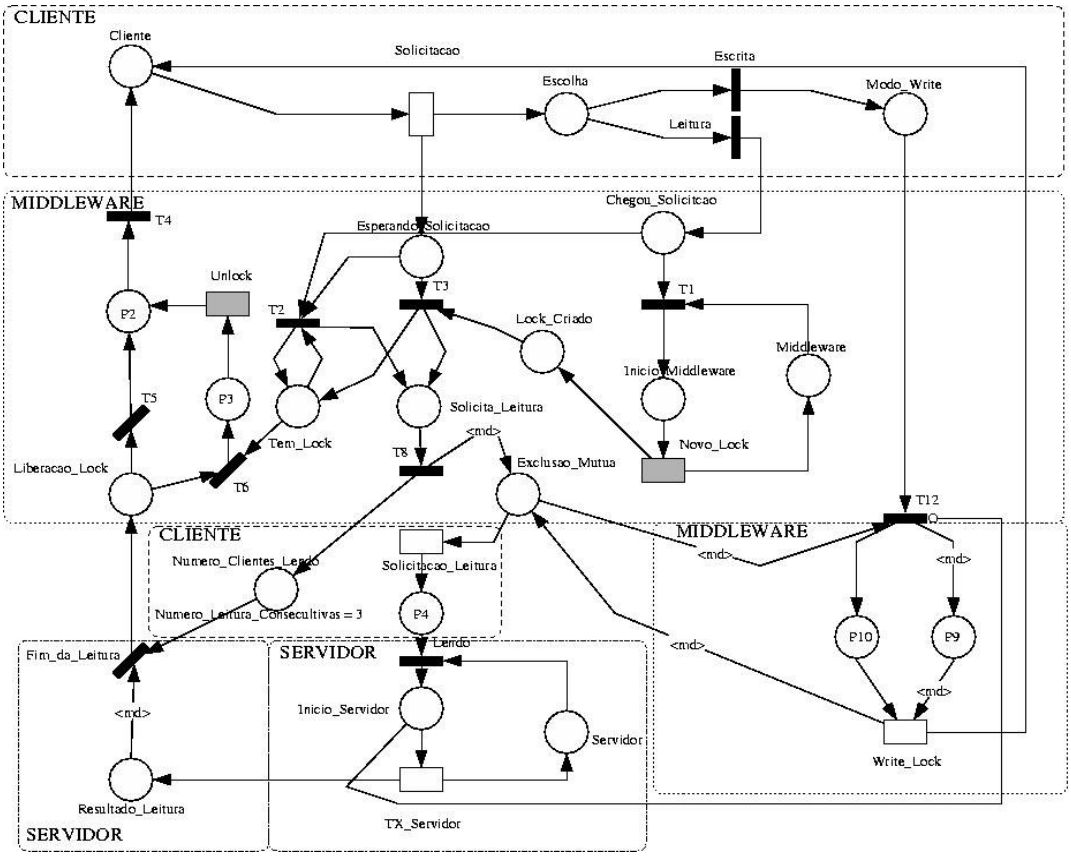


Figura 7-1- Modelo Abstrato

O cenário descrito a seguir apresenta, inicialmente, os parâmetros fixos e variáveis do Cliente, do *Middleware* e, do Servidor. Neste cenário, alguns valores são fixos e foram obtidas na fase de Medição, apresentadas no Capítulo 6. Esses

valores são a taxa de solicitação de serviço (1/6,733348 ms^{-1}), atribuída à transição Solicitação; a taxa de criação do *lock* (1/0,0117469 ms^{-1}), atribuída à transição Novo_Lock; e a taxa de liberação do *lock* , atribuída à transição *Unlock* (1/0,00951843 ms^{-1}). A probabilidade de leitura é atribuída à transição Leitura, definida como 90% e a probabilidade de escrita é atribuída à transição Escrita, definida como 10%. Adota-se que cada cliente realiza três leituras consecutivas, representadas pela variável Número_Leituras_Consecutivas, definida como 3, que rotula o peso dos arcos de saída da transição T8 ao lugar Exclusão_Mútua, os arcos de saída da transição Escrita ao lugar Exclusão_Mútua, os arcos de saída do lugar Exclusão_Mútua a transição T12 e também os arcos de saída do lugar Resultado_Leitura. Assim, com apenas um *lock* no modo de leitura, um cliente realiza 3 leituras consecutivas.

Assume-se, portanto, que o *middleware* pode atender até 1000 clientes simultaneamente[39]. De fato, um número maior de clientes poderia ser utilizado. O número adotado nesse cenário não apresenta uma restrição para o desempenho do sistema. O número máximo de clientes é representado pela marcação do lugar Capacidade_*Middleware* (definida como 1000, indicando que no máximo 1000 *locks* podem ser criados simultaneamente). A probabilidade de escolha entre liberar e reter o *lock* é atribuída às transições T5 e T6, respectivamente. Desta forma, T5 representa a retenção do *lock*, definida como 10%; e T6 a liberação do *lock*, definida como 90%.

Adota-se ainda nesse cenário, que a taxa de solicitação de leitura, atribuída à transição Solicitação_Leitura, é igual à taxa do servidor, atribuída à transição Servindo e definida como (0,5 ms^{-1}). Ou seja, a cada 2 ms^{-1} ocorre uma nova solicitação de leitura e o tempo para realização da leitura é igual a 2 ms^{-1}. O servidor possui uma capacidade de atendimento, atribuída ao lugar Capacidade_Servidor, definida como 10, indicando que 10 clientes podem ser servidos simultaneamente. O tempo médio de escrita é de 2 milissegundos e obedece a uma função de distribuição exponencial (atribuída à transição Write_Lock). Os valores de criação e liberação do *lock* são acumulados em uma única transição (exponencial), pois o foco de estudo dessa dissertação é o modo

de leitura. A análise estacionária foi realizada, considerando o cenário acima descrito e os resultados obtidos são apresentados nas Figuras 7.2, 7.3, 7.4 e 7.5. Devido a memória disponível da máquina utilizada, só foi possível realizar o experimento da Figura 7.2 e 7.4 até 7 clientes e da Figura 7.3 e .5 apenas 5 clientes.

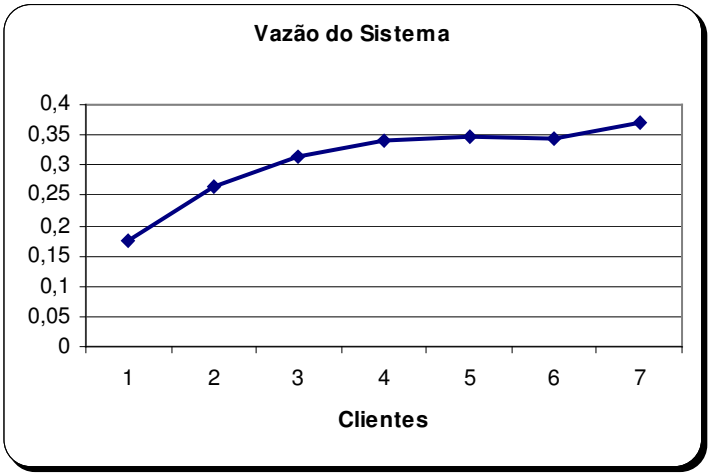


Figura 7-2- Vazão do Sistema com a Técnica de Análise.

Na Figura 7.2, observa-se que a vazão do sistema varia de 0,17 a 0,37ms^{-1}, pois, à medida que se aumenta o número de clientes, aumenta-se o número de clientes solicitando uma nova leitura. Considerando 7 clientes, uma operação de leitura é completada a cada 2,68 ms.

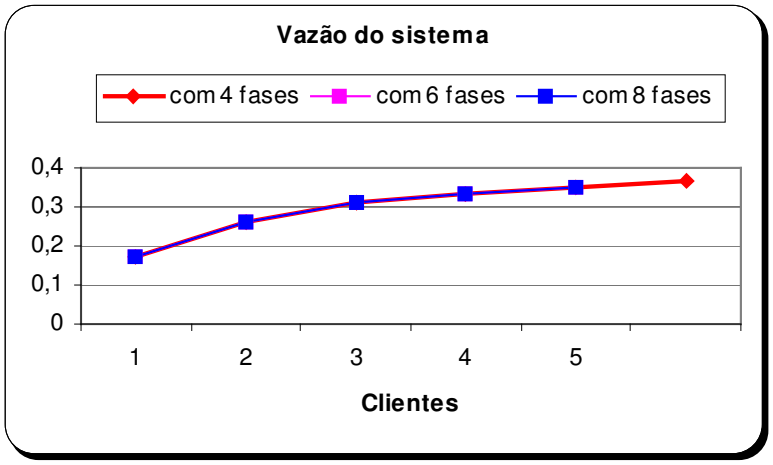


Figura 7-3- Vazão do Sistema com Erlang.

A Figura 7.3 apresenta os resultados obtidos em relação à vazão do sistema. Para isso, utiliza-se o modelo refinado em relação aos tempos de criação e liberação do *lock*, utilizando uma sub-rede Erlang (ver Figura 6.7). Considerando os refinamentos com 4, 6, e 8 fases, observa-se que as curvas ficaram superpostas. Percebe-se que a vazão do sistema varia de 0,17 a 0,36 ms^{-1}, pois, à medida que se aumenta o número de clientes, o número de clientes solicitando uma nova leitura cresce. Desta forma, considerando 5 clientes, a cada 2,74 ms uma leitura é realizada.

Devido à restrição encontrada, só foi possível utilizando a técnica de análise estacionária considerar um número reduzido de clientes. Portanto, verificou-se que os valores apresentados na Figura 7.3 (5 clientes), para calcular a vazão do sistema utilizando sub-rede *Erlang* com 4, 6 e 8 fases, aproximaram-se dos valores apresentados na Figura 7.2 (7 clientes), utilizando transição exponencial.

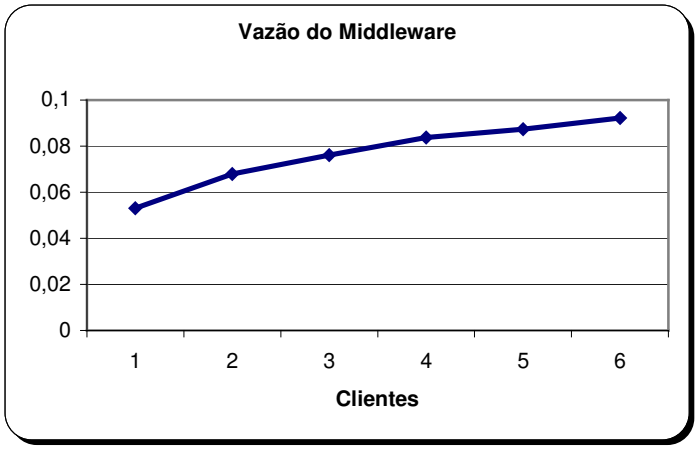


Figura 7-4- Vazão do *Middleware* com a Técnica de Análise.

A Figura 7.4 se observa que a vazão do *middleware* tende a aumentar, variando de 0,053 a 0,10 ms^{-1}, devido ao crescimento do número de clientes que solicitam *locks*. Conseqüentemente, a quantidade de *locks* criados por milessundo também aumenta. Portanto, considerando 7 clientes solicitando *locks*, a cada 10,2 milissegundos, um novo *lock* é criado para que uma nova leitura seja realizada.

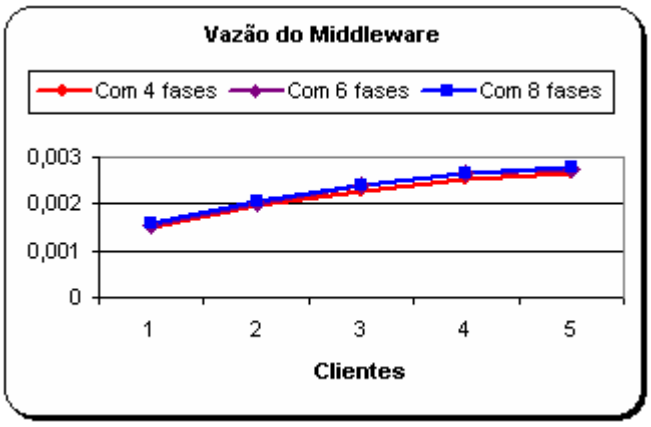


Figura 7-5- Vazão do *Middleware* com Erlang

A Figura 7.5 apresenta os resultados obtidos em relação à vazão do *middleware*. Para isso, utiliza-se o modelo refinado em relação aos tempos de criação e liberação do *lock*, utilizando uma sub-rede Erlang (ver Figura 6.7). Observa-se que as curvas ficaram superpostas, considerando-se os refinamentos com 4, 6, e 8 fases. Percebe-se que a vazão do *middleware* tende a aumentar, variando de 0,015 a 0,027 ms^{-1}, devido ao crescimento do número de clientes que

solicitam *locks*. Conseqüentemente, a quantidade de *locks* criados por milesegundo também aumenta. Desta forma, considerando 5 clientes solicitando *locks*, a cada 37,37 ms, um novo *lock* é criado.

Os resultados descritos na Figura 7.4 não se aproximam dos apresentados na Figura 7.5, devido a maior variância das variáveis aleatórias (tempos) do modelo considerado (aproximação exponencial), porém a tendência de crescimento é apresentada pelos dois modelos. Devido, a restrição de memória da máquina utilizada, só foi possível considerar 7 clientes na utilização da transição exponencial (ver Figura 6.6) e 5 clientes na utilização da sub-rede Erlang.

Para os demais experimentos, utilizou-se como técnica de avaliação a simulação estacionária, devido à restrição de memória da máquina utilizada e a necessidade de avaliar o desempenho considerando um número maior de clientes.

7.3. SEGUNDO EXPERIMENTO

Neste experimento, utiliza-se o cenário descrito na Seção 7.1.1. A métrica avaliada é a vazão do sistema e do *middleware* em função do número de clientes. Utilizando os modelos refinados apresentados na Seção 6.4 e a técnica de avaliação de desempenho utilizada foi a simulação estacionária. Os resultados são apresentados nas Figuras 7.6 e 7.7.

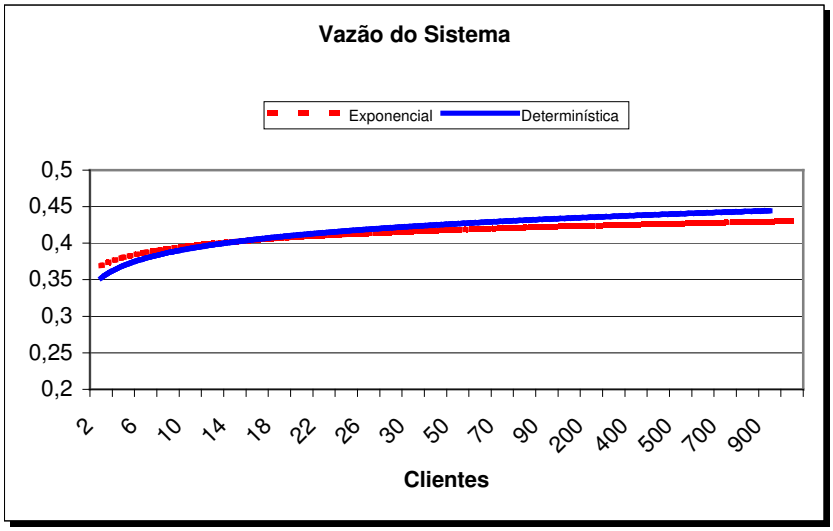


Figura 7-6- Vazão do Sistema com a Técnica de Simulação versus Clientes.

Na Figura 7.6, observa-se que a vazão do sistema aumenta. Considerando 7 clientes a vazão do sistema fica em torno de 0,4 por milessegundo, ou seja, a cada 2,5 ms uma leitura é realizada. Entretanto na análise estacionária tem-se um tempo médio em torno de 2,56 ms. Desta forma, percebeu-se que a técnica de simulação estacionária utilizada na Figura 7.6 apresenta valores próximos dos apresentados nas Figuras 7.2 e 7.3 que utilizam a técnica de análise estacionária.

As curvas mostram que o crescimento do número de clientes para o cálculo da vazão do sistema utilizando transição determinística, representada pela curva contínua, varia de 0,356881 a 0,413739 e utilizando transição exponencial, representada pela curva pontilhada, varia de 0,368352 a 0,427738. Percebe-se que a vazão do sistema aumenta e depois tende a estabilizar, pois existem vários gargalos, como a taxa de solicitação de leitura e a capacidade do servidor. Já o tempo médio diminui, no modelo utilizando transição determinística, varia de 2,85 ms a 2,43 ms, e no modelo utilizando transição exponencial varia de 2,7 ms a 2,38 ms.

Portanto, a variação do número de clientes influencia a vazão do sistema, e como analisa-se o desempenho, a transição exponencial é mais indicada, porque possui menor tempo para realizar a operação de leitura.

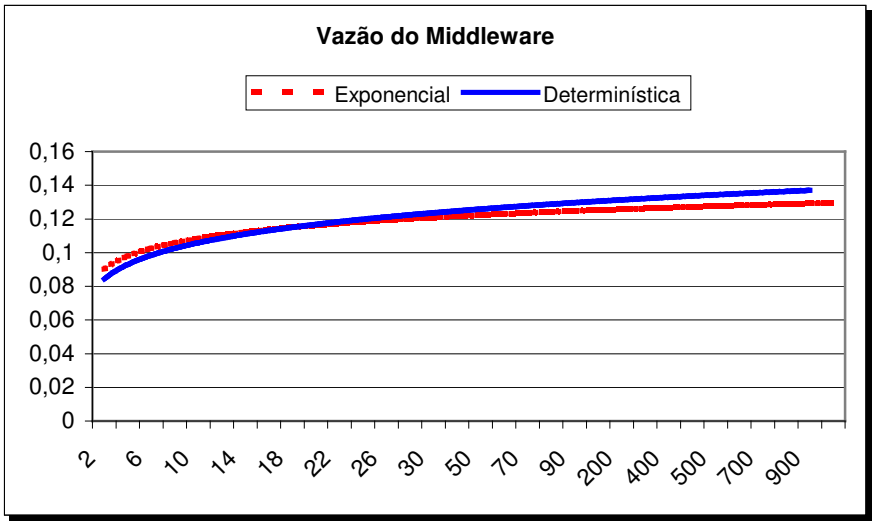


Figura 7-7 - Vazão do Middleware com a Técnica de Simulação versus Clientes.

Na Figura 7.7, observa-se que a vazão do *middleware* aumenta. Considerando 6 clientes, a vazão do *middleware* fica em torno de 0,1 ms^{-1}.Ou seja, a cada 10 ms, um *lock* é criado. Na Figura 7.4, o tempo de criação de um *lock* é igual a 10,5 ms. Assim, ao utilizar-se a técnica de simulação estacionária, obtêm-se valores próximos aos da técnica de análise estacionária.

As curvas mostram que, variando-se a quantidade de clientes, a vazão do *middleware* utilizando transição determinística, representada pela curva contínua, varia de 0,0663889 a 0,124604 e. utilizando transição exponencial, representada pela curva pontilhada, varia de 0,0703402 a 0,127753. Com esses resultados, observa-se que nos dois modelos a vazão do *middleware* aumenta e depois tende a estabilizar. Portanto, a estabilidade de ambas as curvas, deve-se ao fato de que, o *middleware* possui uma capacidade elevada em relação ao número de clientes que solicitam a criação de um *lock*. Sugere-se que aumenta a quantidade de clientes que solicitam a criação de *locks*, pois o *Middleware* possue um a capacidade superior ao número de clientes definido neste cenário.

Para os demais experimentos, utilizou-se como técnica de avaliação a simulação estacionária, devido à restrição de memória encontrada na utilização da análise estacionária.

7.4. TERCEIRO EXPERIMENTO

Neste experimento a métrica avaliada foi à vazão do sistema e do *middleware* em função da probabilidade de leitura (10% a 90%). O cenário utilizado na seção 7.2, apenas um novo parâmetro é definido. O número de clientes atribuído ao lugar Cliente foi 50, indicando que existem 50 clientes fazendo solicitações de um serviço e os resultados obtidos, neste experimento, são apresentados nas Figuras 7.8 e 7.9.

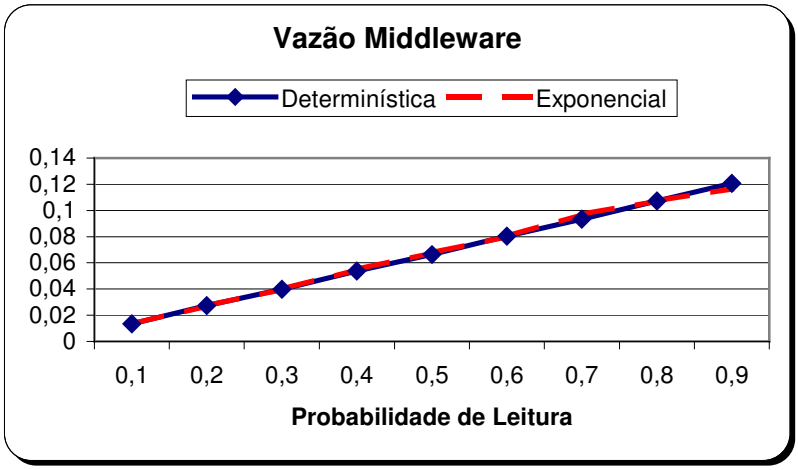


Figura 7-8 - Vazão do *Middleware* versus Probabilidade de Leitura

Na Figura 7.8, observa-se que a vazão do *middleware* tende a aumentar à medida que cresce a probabilidade de criação do *locks*. Assim, quando a probabilidade de leitura é baixa, tem-se um número menor de solicitações ao *middleware*, conseqüentemente, o número de *locks* criados é menor.

Nesse gráfico, o cálculo da vazão do *middleware*, utilizando transição determinística representada pela reta contínua, varia de 0,0134788 a 0,120651 ms^{-1} e, utilizando transição exponencial, representada pela reta pontilhada, varia de 0,0131042 a 0,116607 ms^{-1}. Com esses resultados, observa-se que nos dois modelos utilizados a vazão do *middleware* aumenta, pois o *middleware* possui um capacidade elevada, e um número maior de clientes poderia ser utilizado. Sugere-se que, para uma análise apenas das operações de leituras, a probabilidade de leitura seja definida como 0,9.

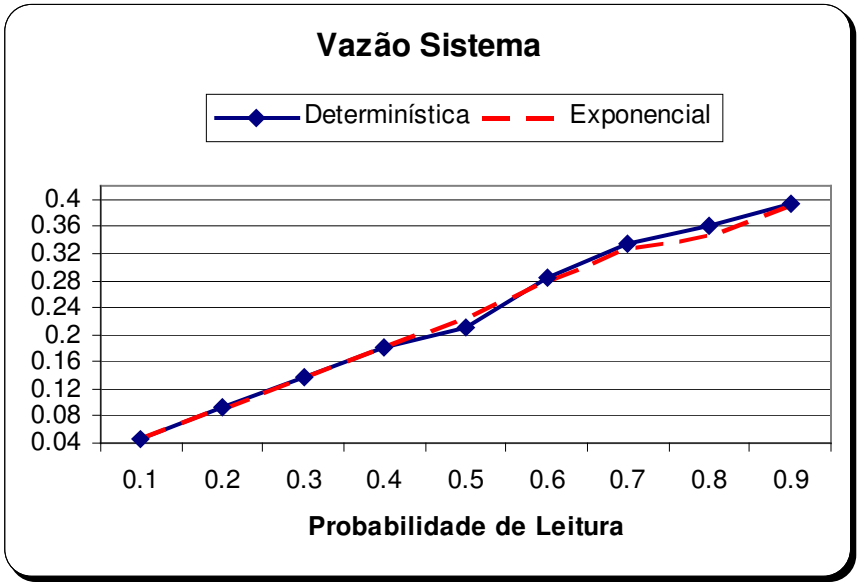


Figura 7-9- Vazão do Sistema versus Probabilidade de Leitura.

Na Figura 7.9, observa-se que a vazão do sistema tende a aumentar à medida que aumenta a probabilidade do modo de leitura. Com isso, aumenta também o número de clientes solicitando uma leitura e o número de clientes que realizam uma leitura completa (solicita e recebe a resposta).

Nesse gráfico, a vazão do sistema utilizando transição determinística, representada pela curva contínua, varia de 0,0471776 a 0,392984 ms^{-1} e, utilizando transição exponencial, representada pela curva pontilhada, varia de 0,0473628 a 0,390049 ms^{-1}. Com esses resultados, observa-se que nos dois modelos a vazão do sistema aumenta, Assim, as taxas pré-definidas (Solicitação_Leitura e Leitura) não representam restrição de desempenho.

Nesse caso, a diferença dos resultados providos por ambos os modelos utilizando essas transições é da ordem de centésimos de milessegundo. Assim, a variação da probabilidade de leitura influencia a vazão do sistema, pois o crescimento da probabilidade de leitura aumenta a quantidade de clientes que desejam realizar uma operação de leitura no servidor, aumentando a vazão do sistema.

7.5. QUARTO EXPERIMENTO

Neste experimento, as métricas avaliadas foram, a capacidade do servidor, a vazão do sistema e do *middleware* em função da taxa de solicitação de leitura. O cenário utilizado apresenta as características descritas anteriormente (ver seção 7.2) e apenas a capacidade do servidor foi redefinida como 15, indicando que o servidor pode servir até 15 solicitações simultaneamente. Os resultados encontrados são representados graficamente nas Figuras 7.10, 7.11 e 7.12.

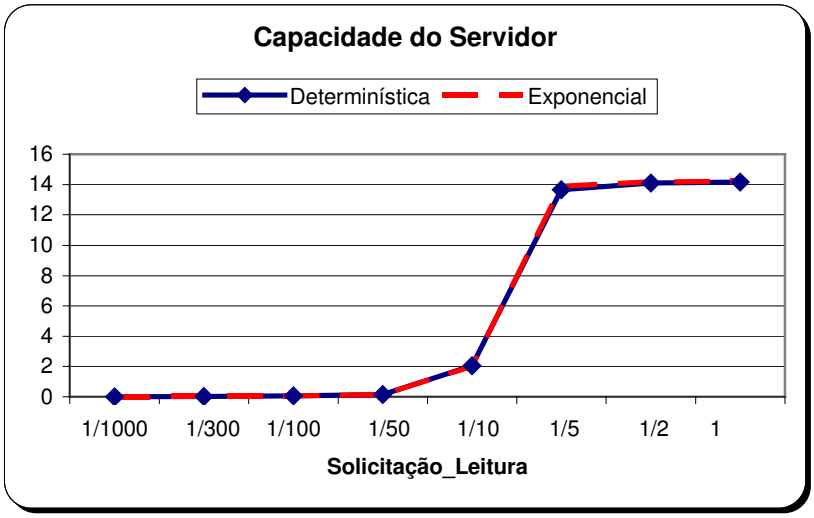


Figura 7-10- Capacidade do servidor versus Taxa de Solicita;áo de Leitura.

Na Figura 7.10, verifica-se que, aumentando a taxa de solicitação de leitura, aumenta-se o número de clientes sendo servidos até o limite do servidor, que é 15. A partir desse ponto, obtém-se a saturação do servidor. Os resultados obtidos pela avaliação dos modelos com transições deterministica e transições exponencias variam de 0,006771 a 14,2009 ms^{-1} e 0,006771 a 14,2009 ms^{-1}, respectivamente.Com estes resultados, observa-se que o valor adotado como capacidade do servidor limita o número médio de clientes sendo servidos. Assim, quando a taxa de solicitação de leitura está em torno de 6 ms^{-1}, o servidor chega ao seu limite, portanto, uma possível recomendação seria aumentar a capacidade do servidor para que sejam atendidas maiores cargas de leitura.

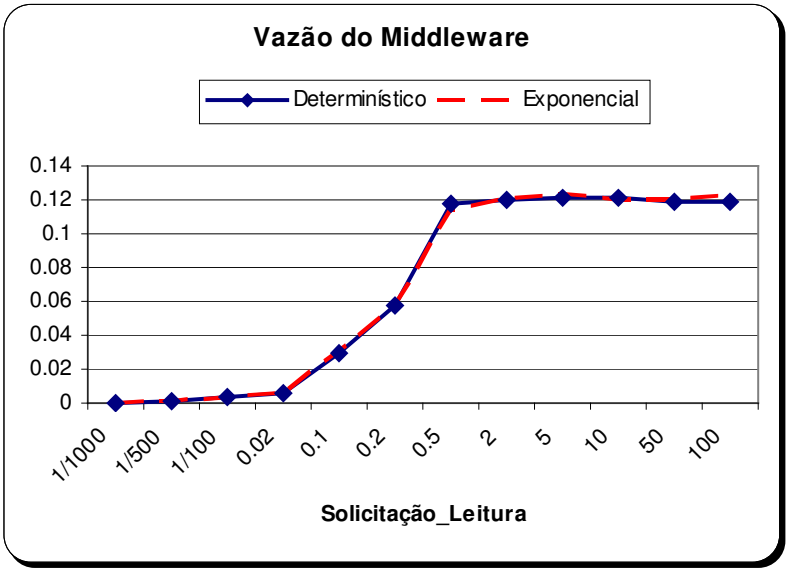


Figura 7-11- Vazão do *Middleware* versus Taxa de Solicitação de Leitura.

Na Figura 7.11, observa-se que a vazão do *middleware* aumenta, à medida que aumenta a taxa de solicitação de leitura. Com isso, aumenta-se o número de clientes que criam *lock*, pois a probabilidade de liberar o *lock* está definida como 90%.

Nestas curvas, a vazão do *middlewar*ei calculada através do modelo com utilizando transição determinística, representada pela curva contínua, varia de 0,000303 a 0,118537 ms^{-1} e o re sultado obtido atrvés do modelo com utilizando transição exponencial, representada pela curva pontilhada, varia de 0,000309 a 0,122045 ms^{-1}. Com esses resultados, observa-se que, nos dois modelos. a vazão do *middleware* aumenta, conseqüentemente as solicitações de leitura tornam-se mais frequentes, chegando a um ponto de estabilidade. Pois a capacidade do servidor (10) está limita o número de operações de leitura, conseqüentemente os *locks* não serão liberados e o *middleware* também tenderá a estabilidade.

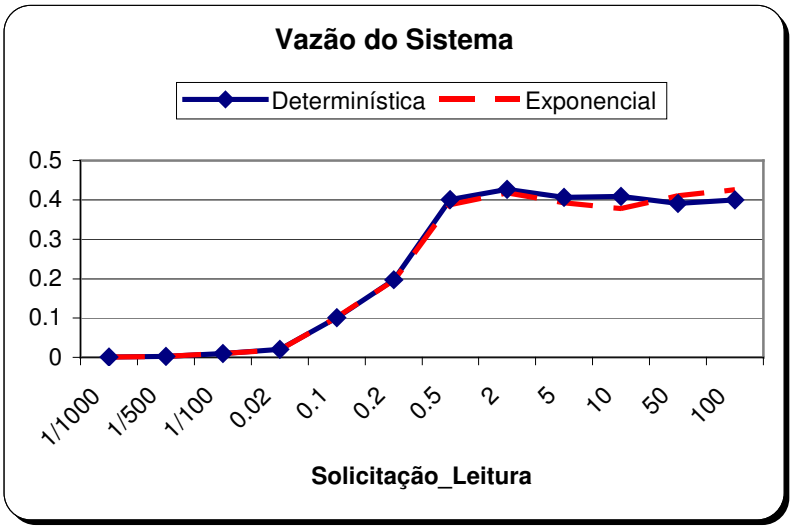


Figura 7-12 - Vazão do Sistema versus Taxa de Solicitação de Leitura.

Na Figura 7.12, observa-se que a vazão do sistema tende a aumentar à medida que varia à taxa de solicitação de leitura, até estabilizar.

Neste gráfico, a vazão do sistema calculada através do modelo com transições determinísticas, varia entre 0,00104 a 0,400326 ms^{-1} e os valores obtidos através do modelo com transições exponenciais, varia de entre 0,00104 a 0,426033 ms^{-1}. Como nos casos anteriores (ver Figura 7.10 e 7.11), a capacidade do servidor limita o sistema, desta forma, mesmo que a taxa de solicitação de leitura aumente, o servidor não tem capacidade de atender mais solicitações.

7.6. QUINTO EXPERIMENTO

Este experimento consiste na avaliação da vazão do sistema e do *middleware* em função da taxa de liberação da operação *lock* (10% a 90%). O cenário descrito a seguir, inicialmente, os parâmetros fixos e variáveis em relação os componentes do sistema(Cliente, *Middleware* e Servidor) definidos na seção 7.2. Apenas o número de clientes, atribuído ao lugar Clientes, é definido como 30 e os resultados das métricas são representados através das Figuras 7.13 e 7.14.

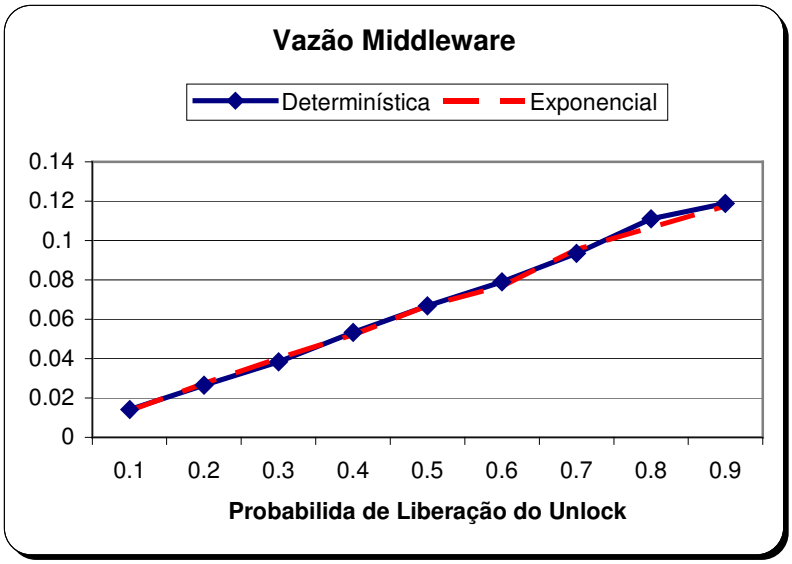


Figura 7-13 - Vazão do *Middleware* versus Probabilidade de Liberação do *lock*.

Na Figura 7.13, observa-se que a vazão do *middleware* aumenta, à medida que aumenta a probabilidade da taxa de liberação do *lock*, conseqüentemente o número de clientes que solicitam a criação de um novo *lock* também aumenta.

Nesse gráfico, a vazão do *middleware*, utilizando transição determinística, representado pela curva contínua, varia de 0,0141182 a 0,118882 ms^{-1} e, utilizando transição exponencial, representado pela curva pontilhada, varia de 0,0135249 a 0,117409 ms^{-1}. Com esses resultados, verifica-se que o *middleware* passa a ser mais utilizado quando se aumenta a probabilidade dos clientes liberam o *lock*. Assim, criações de *locks* são solicitadas com maior freqüência e o *middleware* possui capacidade para atender essas solicitações. Sugere-se aumento do número de clientes.

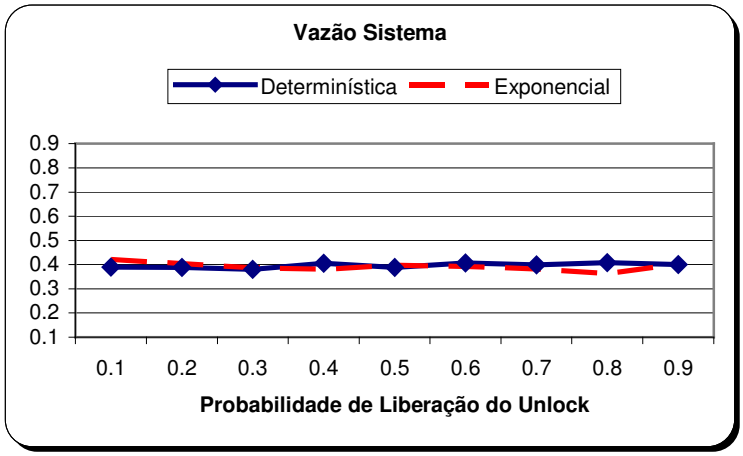


Figura 7-14 - Vazão do Sistema versus Taxa de Liberação do *Lock*.

Na Figura 7.14, observa-se que a vazão do sistema tende a estabilizar, pois, à medida que a probabilidade da taxa de liberação do *lock* aumenta. Nesse gráfico, a vazão do sistema, utilizando transição determinística, representado pela curva contínua, varia de 0,389039 a 0,39927 ms^{-1} e, utilizando transição exponencial, representada pela curva pontilhada, varia de 0,42124 a 0,402305 ms^{-1}. Percebe-se que a variação da taxa de liberação do *lock* tem pouca influência na vazão do sistema, pois, anteriormente, a liberação do *lock*, a operação de leitura é realizada.

7.7. SEXTO EXPERIMENTO

Neste experimento, as métricas calculadas foram: capacidade do servidor, vazão do sistema e do *middleware* em função da variação da taxa do servidor. Neste cenário alguns parâmetros fixos e variáveis do Cliente, do *Middleware* e do Servidor descritos na seção 7.2 foram alterados para realização destas métricas. Assim, o número de clientes, atribuído ao lugar Cliente, é definido como 30. Adota-se que a taxa de solicitação de leitura, atribuída à transição Solicitação_Leitura, é definida como 6 ms. O servidor possui uma capacidade de atendimento, atribuída ao lugar Capacidade_Servidor igual a 15. Os resultados obtidos são apresentados nas Figuras 7.15, 7.16, 7.17.

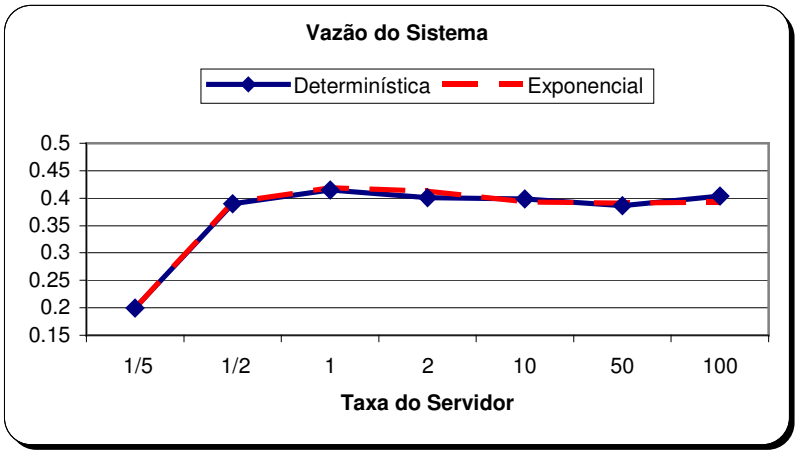


Figura 7-15 – Vazão do Sistema versus Taxa do Servidor(Servindo)

Na Figura 7.15, observa-se que a vazão do sistema, aumenta e depois estabiliza. Neste gráfico, a vazão do sistema utilizando transição determinística, representado pela curva contínua, varia de 0,199324 a 0,403929 ms^{-1} e, utilizando transição exponencial, representado pela curva pontilhada, varia de 0,198978 a 0,393091 ms^{-1}. Desta forma, observa-se que mesmo tornando o servidor mais rápido a vazão do sistema (Taxa do Servidor > ½) se estabiliza, indicando, por exemplo, que a carga do cenário não demanda maior poder computacional do servidor.

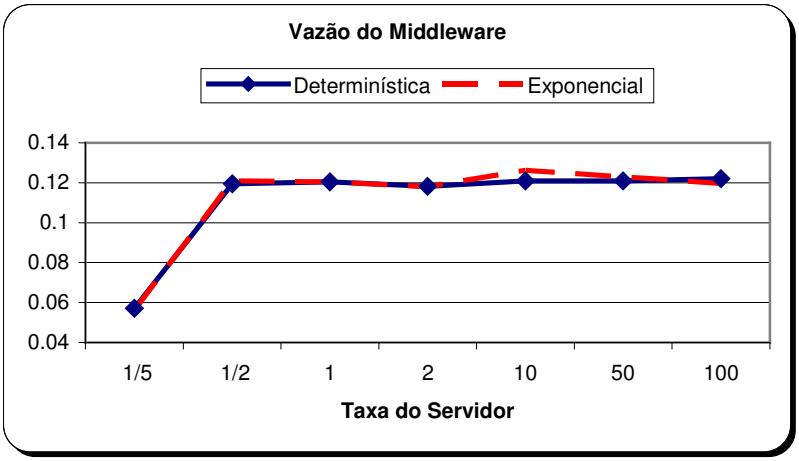


Figura 7-16 - Vazão do *Middleware* versus Taxa do Servidor(Servindo).

Na Figura 7.16, observa-se que a vazão do *middleware* aumenta e depois estabiliza. No ponto em que a vazão do *middleware* começa a estabilizar, um *lock*

é criado a cada 0,5 ms. Nesse gráfico, a vazão do sistema utilizando transição determinística varia entre 0,057112 a 0,12207 ms^{-1} e, utilizando transição exponencial, varia entre 0,056072 a 0,119635 ms^{-1}. Portanto, o *middleware* tem capacidade de criar mais *locks*, possibilitando o incremento do número de clientes e/ou das taxas de solicitação de leitura/serviço.

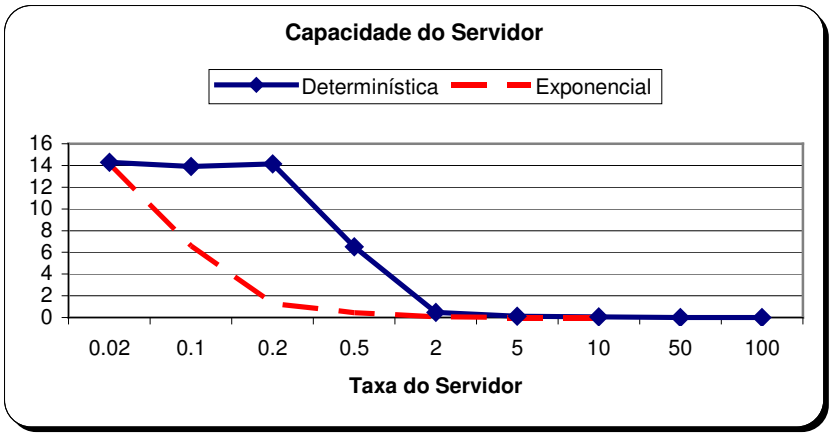


Figura 7-17 - Capacidade do Servidor versus Taxa Servidor (Servindo).

Na Figura 7.17, observa-se que a capacidade do servidor diminui à medida que a taxa do servidor aumenta, pois, normalmente quando se aumenta esta taxa, o tempo médio diminui.

Nesse gráfico, a capacidade do servidor, utilizando transição determinística, varia de 14,15606 a 0,0042 ms^{-1} e, utilizando transição exponencial, varia de 14,11138 a 0,004068 ms^{-1}. Com esses resultados, a capacidade do servidor é suficiente para atender à demanda dos clientes. Assim, todos os clientes que solicitam leituras, são atendidos.

7.8. CONSIDERAÇÕES FINAIS

Neste capítulo cenários foram definidos e seis experimentos foram realizados. Nesses experimentos compararam-se as técnicas de avaliação de desempenho e contatou-se que a técnica de avaliação baseada em análise estacionária foi inadequada, devido à quantidade de memória do computador utilizado. Assim, a simulação foi à técnica adotada nos experimentos posteriores

(segundo, terceiro, quarto, quinto e sexto), os modelos (transição exponencial e transição determinística) foram utilizados. Posteriormente, os resultados foram interpretados, e possíveis recomendações foram sugeridas.

8. CONCLUSÕES E TRABALHOS FUTUROS

Este capítulo apresenta as conclusões do trabalho realizado, ressaltando as contribuições sobre avaliação de desempenho do SCC do CORBA. Por fim, são apresentadas também sugestões de trabalhos para dar continuidade ao estudo do desempenho de middleware.

O processo de avaliação baseada em modelos possibilita a análise de desempenho de sistemas, antes mesmo de sua implementação, o que possibilita ajustes ainda na fase de desenvolvimento. O uso de modelos como mecanismo de avaliação também torna possível a avaliação de cenários complexos, possibilitando, portanto a análise de desempenho em função de restrições temporais e de recursos. Para construir esses modelos, o formalismo de redes de Petri Redes Estocásticas foi adotado.

Este trabalho propôs modelos redes de Petri estocástica para avaliar o desempenho do SCC do CORBA. Para isso, uma metodologia de avaliação de desempenho foi proposta, com a finalidade de definir o sistema avaliado e seus componentes, definir métricas para a medição, construir modelos, validar os modelos refinados através da mensuração dos resultados da avaliação das redes de Petri com as métricas mensuradas no SCC do CORBA. Um conjunto de experimentos foi realizado, seus resultados interpretados e possíveis recomendações sugeridas.

No decorrer deste trabalho, a metodologia proposta possibilitou uma abordagem sistemática, onde foi possível construir um modelo inicial a partir da fase de definição do sistema e seus componentes. Na fase de validação duas técnicas de avaliação de desempenho (Simulação e Análise) foram usadas. Conclui-se que a técnica de simulação poderia ser utilizada, pois os resultados obtidos entre essas técnicas possuíam tendências semelhantes. Posteriormente, esse modelo foi validado qualitativamente através da verificação das propriedades das Redes de Petri. Na fase de Medição, foi realizado um estudo em relação aos

outliers encontrados nas amostras, para que a massa de dados não fosse interpretada erroneamente. Com isso, calcularam-se as estatísticas e aplicou-se a técnica de aproximação por fase, verificou-se que o número de fases geradas segundo os cálculos foi elevado. Sugeriu-se um número menor de fases, devido à restrição de memória encontrada. Os modelos refinados foram validados quantitativamente através da comparação dos valores da sub-rede Erlang, com transições determinísticas (sugeridas no refinamento) e observou-se que ambas possuem tendências semelhantes. Portanto, um número menor de fases foi utilizado e os dados obtidos representaram com fidelidade o sistema avaliado. Na fase de avaliação e análise dos resultados seis experimentos foram realizados e uma variedade de cenários foram definidos.

A contribuição principal é a proposição de três modelos RdPs para avaliação de desempenho do SCC do CORBA. Na obtenção desses modelos utilizou-se a técnica de aproximação por fase para encontrar as distribuições adequadas, sugerindo-se três alternativas que foram validadas e posteriormente utilizadas para realização da avaliação e obtenção dos resultados das métricas. O estudo confirmou a utilidade e eficiência das Rdps estocásticas, pois auxilia a avaliação de desempenho e facilita a obtenção dos resultados referentes às métricas antes que o sistema esteja implementado. A metodologia de avaliação de desempenho proposta apresenta as seguintes contribuições:

- Fornecimento de uma forma sistemática para abordagem de sistemas de avaliação de desempenho e construção de modelos, os quais podem ser usados para verificação do sistema e sua implementação.
- Representação das características e operações do sistema de maneira uniforme e clara.
- Representação do sistema do nível conceitual ao detalhado, de acordo com a estrutura hierárquica das atividades do sistema, considerando a modularidade, a flexibilidade e a capacidade de expansão.

Os modelos propostos representam o SCC do CORBA e produziram resultados relevantes à área de avaliação de desempenho.

8.1. TRABALHOS FUTUROS

Uma série de trabalhos pode ser vislumbrada em relação ao formalismo das Redes de Petri Estocásticas e a metodologia de avaliação de desempenho apresentada. Como sugestões de trabalhos futuros podem ser citadas:

- Modelagem das operações de *change_mode* e *try_lock* do serviço de controle de concorrência do CORBA, considerando os modos de leitura e escrita, intenção de ler, intenção de escrever e *upgrade*.
- Modelagem das operações de *lock* e *unlock* do serviço de controle de concorrência do CORBA, considerando os modos de intenção de ler, intenção de escrever e *upgrade*.
- Modelagem das operações do serviço de controle de concorrência do CORBA, considerando uma granularidade maior do que a mostrada neste trabalho.
- Modelagem das operações de *lock* e *unlock*, *change_mode, try_lock* do serviço de transação do CORBA, considerando todos os modos.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Performance Analysis, First EEF/Euro Summer School on Trends in Computer. Scienc Berg en Dal, The NetherInds, July 3-7, 2000 Revised Lectures.

[2] Maciel, P. Introdução às redes de Petri e aplicações. UNICAMP, Campinas 1996.

[3] T. Murata. Petri Nets: Properties, Analysis and Applications. Proceeding of The IEEE, 1989.

[4] TimeNet 3.0 em http://pdv.cs.tu-berlin.de/~timenet/overv.html.

[5] Marsan, G Chiola. On Petri Nets with Deterministic and Exponentially Distributed Firing Times. Advances in Petri Nets, vol 266, Lecture Notes in Computer Science, Springer Verlag, Edited by G.Rozenberg, 1987.

[6] M. Ajmone Marsan ,Gianfranco Balbo, Gianni Conte, Susanna Donateili, Giuliana Franceschinis, Modelling With Generalised Stochastic Petri Nets, Università degli studi di Tonno Dipartimento di Informatica.

[7] M. K. Molloy. On the Integration of Delay and Throughput Measures in Distributed Processing Models. PhD. Thesis, UCLA, Los Angeles, CA, 1981.

[8] R. German. Performance Analysis of Communicating Systems - Modeling with Non-Markovian Stochastic Petri Nets. John Wiley and Sons, 2000.

[9] G Bolch, S. Greiner, H. de Meer, K. Trivedi. Queueing Networks and Markov Chains - Modeling and Performance Evaluation with Computer Science Applications. John Wiley and Sons, 1998.

[10] R. Zurawski and M. Zhou. Petri nets and industrial applications: a tutorial. In IEEE Transactions on Industrial Electronics, 1994

[11] Object Management Group. The Common Object Request Broker: Architecture and Specification, October 1999.

[12] Boudewijn R. Haverkort, Markovian Models for Performance and Dependability Evaluation,Laboratory for Performance Evaluation and Distributed Systems, Department of Computer Science, RWTH Aachen, 52056 Aachen, Germany

[13] Marco Ajmone Marsan, Gianni Conte, Gianfranco Balbo, A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems, Politecnico di Torino, Universita' di Torino, Turin, Italy.

[14] Steve Vinosk. Where is Middleware? IEEE Internet Computing, 2002.

[15] Alan A. Desrochers and Robert Y. Al-Jaar, Applicatins of Petri Nets in Manufacturing Systems Modeling, Control and Performance Analysis, 1994.

[16] T. S.F.L. Fernandes, W.J. Silva, M. Silva, N.S. Rosa, P.R.M. Maciel, and D.F.H. Sadok. On the generalised stochastic petri net modeling of message-oriented middleware systems. In: 23rd IEEE International performance

Computing and Communications Conference - International Workshop on Middleware Performance (IWMP 2004), 2004.

[17] Oliver C. Ibe, Hoon Choi e Kishor S. Trivedi, Performance Evaluation Client Server System, November, 1993.

[18] TRIOLA, Mário F. Introdução à estatística. (Tradução Vera Regina de Farias e Flores; revisão técnica Ana Maria Lima de Farias e Flores). Rio de Janeiro: LTC, 2005.

[19] [59] B.R. Haverkort and K.S. Trivedi. Specification and generation of Markov reward models. Discrete-Event Dynamic Systems: Theory and Applications, 1993.

[20] C. A. Petri. Comunication with automata, New York, 1966.

[21] [67] Christos G. Cassandras, Stéphane Lafortune, Introduction to Discrete Event Systems, Kluwer Academic Publishers.

[22] W. J. Stewart. Introduction to the numerical solution of Markov Chains. Princeton University Press, 1994.

[23] Christos G. Cassandras, Stéphane Lafortune, Introduction to Discrete Event Systems, Kluwer Academic Publishers A. Jensen. Markov chains as an aid in the study of Markov processes. Skand. Aktuarietidskrift, 1953.

[24] S. Ramani, K. Trivedi, B. Dasrathy Performance Analysis of the CORBA Event Service Using Stochastic Reward Nets. 19th IEEE Symposium on Reliable Distributed Systems, Nurberg, Germany, October, 2000.

[25] G. Ciardo, A. Blakemore, P. F. Chimento Jr, J. K. Muppala, and K. S. Trivedi. Automated generation and analysisof Markov reward models using stochastic reward nets. In C. Meyer and R. Plemmons, editors, Linear Algebra, Markov Chains, and Queueing Models. Springer-Verlag, Heidelberg, 1992.

[26] Ulrich Herzog, Formal Methods for Performance Evaluation, Universität Erlangem-Nürnberg, Institut für informatik, Germany.

[27] Jain, R., The Art of Computer Systems. Techniques for Experimental Design, Measurement, Simulation, and Modeling. John Wiley & Sons, 1991.

[28] W.K. Grassmann. Finding transient solutions in Markovian event systems through randomization. In W.J. Stewart, editor, Numerical Solution of Markov Chains,pages 357–371. Marcel Dekker, 1991.

[29] Performance Evaluation of concurrent systems using timed petri nets W. Reisig. A Primer in Petri Net Design. Springer Verlag, 1992.

[30] Ian Gorton and Anna Liu. Evaluating Object Transactional Monitors with OrbixOTM. In International Symposium on Software Engineering for Parallel and Distributed Systems, pages 210-216, Los Angeles, USA, May 1999.

[31] Srinivasan Ramani, S Kishor Trivedi, and Balakrishnan Dasarathy. Performance Analysis of the CORBA Notification Service. In 20th IEEE Symposium on Reliable Distributed Systems (SRDS), October 2001.

[32] B.R. Haverkort and K.S. Trivedi. Specification and generation of Markov reward models. Discrete-Event Dynamic Systems: Theory and Applications, 1993.

[33] Christos G. Cassandras, Stéphane Lafortune, Introduction to Discrete Event Systems, Kluwer Academic Publishers.

[34] Srinivasan Ramani, S Kishor Trivedi, and Balakrishnan Dasarathy. Performance Analysis of the CORBA Event Service Using Stochastic Reward. In 19th IEEE Symposium on Reliable Distributed Systems (SRDS), October 2000,

[35] Performance Evaluation of concurrent systems using timed petri nets W. Reisig. A Primer in Petri Net Design. Springer Verlag, 1992.

[36] B.R. Haverkort and K.S. Trivedi. Specification and generation of Markov reward models. Discrete-Event Dynamic Systems: Theory and Applications, 1993.

[37] Dorina Petriu, Hoda Amer, Shikharesh Majumdar, and Istabrak Abdul-Fatah. Using Analytic Model for Predicting Middleware Performance. In Second International Workshop on Software Performance, pages 189{194, Ottawa, Canada, September 2000.

[38] Dorina Petriu, Hoda Amer, Shikharesh Majumdar, Istabrak Abdull-Fatah, Using Analytic Models for Predicting Middleware Performance.

[39] Menasce A. Daniel, Virgilio A. F. Almeida, Planejamento de Capacidade para Serviços na Web, editora Campus, 2003.

Dissertação de Mestrado apresentada por **Roberta Andrade de Araújo Fagundes** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **"Avaliação de Desempenho do Serviço de Controle de Concorrência do CORBA Usando Redes"**, orientada pelo **Prof. Paulo Romero Martins Maciel** e aprovada pela Banca Examinadora formada pelos professores:

Prof. Djamel Fawzi Hadj Sadok
Centro de Informática / UFPE

Prof. Gledson Elias da Silveira
Departamento de Informática / UFPB

Prof. Paulo Romero Martins Maciel
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 24 de fevereiro de 2006.

Prof. FRANCISCO DE ASSIS TENÓRIO DE CARVALHO
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.