UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

TÉCNICAS PARA AVALIAÇÃO DE DESEMPENHO DE SISTEMAS COMPUTACIONAIS

REGINA HELENA CARLUCCI SANTANA
MARCOS JOSÉ SANTANA
REGINA CÉLIA G. SOUSA ORLANDI
RENATA SPOLON
NIVALDI CALÔNEGO JÚNIOR

Nº 14

NOTAS DIDÁTICAS

São Carlos - SP

UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação
ISSN 0103-2585

TÉCNICAS PARA AVALIAÇÃO DE DESEMPENHO DE SISTEMAS COMPUTACIONAIS

REGINA HELENA CARLUCCI SANTANA
MARCOS JOSÉ SANTANA
REGINA CÉLIA G. SOUSA ORLANDI
RENATA SPOLON
NIVALDI CALÔNEGO JÚNIOR

Nº 14

NOTAS DIDÁTICAS

São Carlos – SP
Set./1994

Universidade de São Paulo
Instituto de Ciências Matemáticas de São Carlos
Departamento de Ciências de Computação e Estatística

Técnicas Para Avaliação de Desempenho de Sistemas Computacionais

Regina Helena Carlucci Santana
Marcos José Santana
Regina Célia G. Sousa Orlandi
Renata Spolon
Nivaldi Calônego Júnior

São Carlos
Setembro de 1994

Técnicas Para Avaliação de Desempenho de Sistemas Computacionais

Capítulo 1

Introdução

Uma figura de desempenho para um sistema computacional corresponde ao resultado de uma análise da quantidade de serviços prestados em relação ao tempo total decorrido desde o início desses serviços. Assim, o desempenho passa a ser o agente que reflete diretamente o andamento de qualquer tarefa iniciada pelo usuário dentro do sistema [FE92]. O desempenho de um sistema computacional é, prioritariamente, observado por seus usuários através dos tempos de respostas experimentados. De certa forma, a qualidade de um sistema é julgada pela impressão causada aos seus usuários, através dos tempos de respostas observados [SA90]. Isso não constitui um parâmetro confiável, mas sempre que existe um certo grau de descontentamento entre um grupo de usuários de um dado sistema, uma avaliação criteriosa deverá ser elaborada a fim de se identificarem possíveis pontos de estrangulamentos [LU71,KO78,SA90].

Alguns possíveis empregos da análise de desempenho em Sistemas Computacionais Distribuídos são: *análise de sistemas existentes*, *análise de sistemas ainda em desenvolvimento* ou na *seleção de um determinado sistema* [LU71,KO78, SA90, SA90a]. Em sistemas existentes seus possíveis objetivos compreendem a maximização da eficiência, maximização da utilização de um dado recurso, processamento de uma carga de trabalho a um custo mínimo, minimização do tempo de resposta, entre outros. Para sistemas inexistentes a avaliação deve ser prognóstica [SA90a], ou seja, deve fazer previsões do comportamento do novo sistema. Dessa forma a necessidade de avaliação de desempenho e prognóstico existem desde a concepção inicial da arquitetura do sistema até sua operação diária, após sua instalação. Para a seleção de um novo sistema é feita um comparação entre os desempenhos desses sistemas e a opção é feita segundo um parâmetro específico. As técnicas usadas para a avaliação de desempenho e prognóstico durante essas fases vão de simples cálculo manual a soluções de equações muito complexas e/ou simulações extremamente sofisticadas.

Apesar da importância de se efetuar a análise de desempenho, muitas vezes essa tarefa é desprezada e freqüentemente evitada, com conseqüências drásticas para as atividades do sistema em questão. Esse desuso, na maioria dos casos, é atribuído a dois fatores: os interessados não dominam o emprego de ferramentas para efetuar a análise, ou então (na maioria dos casos) a avaliação torna-se uma tarefa difícil, apesar de se ter domínio sobre a aplicação dessas ferramentas. De modo geral, dependendo do tipo de avaliação desejada e do tipo de sistema em consideração, é possível o

envolvimento com ferramentas teóricas (modelamento analítico do sistema e solução das equações obtidas) [KO78], simulação (desenvolvimento e implementação de um modelo de simulação que represente com fidelidade o sistema real) [DA89, MA87, SA90, SA90a] ou ainda experimentação prática, para sistemas existentes (instrumentação do sistema para coleta de dados e posterior análise) [KO78].

1.1 Medidas de Desempenho

Afirmar que o desempenho de um sistema é ótimo significa que a qualidade do serviço oferecido pelo sistema excede alguma expectativa. Mas a "medida da qualidade do serviço" e "exceder a expectativa" variam dependendo dos indivíduos envolvidos, projetistas do sistema, gerentes de instalação ou usuários de terminais [WE87].

Visando agrupar os indivíduos com diferentes interesses, podem-se classificar as medidas de desempenho em duas categorias: "medidas orientadas ao usuário" e "medidas orientadas ao sistema" [KO78].

As medidas orientadas ao usuário incluem quantidades como tempo de ciclo ("turnaround time") em um ambiente baseado em processamento de lotes e o tempo de resposta em um sistema de tempo real e/ou interativo. O tempo de ciclo corresponde ao intervalo de tempo decorrido desde a requisição da tarefa até a disponibilidade do resultado processado. Analogamente, em um ambiente interativo o tempo de resposta de um pedido representa o intervalo decorrido desde a chegada do pedido até que seja completado. Geralmente os pedidos são colocados em classes de prioridades diferentes. Dessa forma associa-se a um pedido individual o valor de prioridade de sua classe, conhecendo-se, assim, sua urgência, sua importância e as características de demanda do recurso. O tempo de ciclo e o tempo de resposta, usualmente, são definidos e calculados separadamente para cada classe de prioridade.

As medidas orientadas ao sistema são tipicamente a capacidade de trabalho ("throughput") e a utilização. A capacidade de trabalho corresponde ao número médio de pedidos processados por unidade de tempo - avalia o grau de produtividade que o sistema pode fornecer. Geralmente a carga do sistema é expressa em função da proporção de carga máxima que o sistema pode controlar. Essa proporção é a utilização do sistema que pode ser definida pela razão entre a carga real do sistema pela carga máxima que o sistema pode controlar. Por exemplo, em termos dos recursos do sistema tem-se que a utilização corresponde à fração de tempo que o recurso permaneceu ocupado.

1.2 Técnica Para Análise de Desempenho

Durante a avaliação de desempenho de um sistema o avaliador deve colher informações associadas aos parâmetros significativos à análise, para uma dada carga de trabalho [FE92]. Técnicas de avaliação são os métodos pelos quais tais informações podem ser obtidas. As informações requeridas por um estudo podem ser obtidas a partir do próprio sistema (técnicas de aferição - "benchmarking", coleta de dados e construção de protótipos) ou então através de um modelo representativo do sistema (modelo analítico ou modelo de simulação).

As técnicas de modelagem são, em geral, adotadas uma vez que os sistemas computacionais modernos são complexos e as técnicas de aferição, constituem tarefas que não são fáceis de serem conduzidas [SA90]. Além disso, essas técnicas devem ser empregadas quando se deseja analisar o desempenho de um sistema já implementado (ou em fase final de desenvolvimento).

Técnicas de avaliação baseadas em modelos requerem que esses modelos sejam resolvidos. A solução pode ser obtida através de métodos analíticos ou de simulações [KO78, SA90]. A confiabilidade de um modelo pode ser assegurada com o conhecimento prévio de que sua base conceitual esteja correta. Sem essa certeza pode-se apenas concluir que os resultados provenientes do modelo refletem as tendências do comportamento do sistema real.

Algumas técnicas relevantes na avaliação de desempenho são mostradas: (figura 1.1)

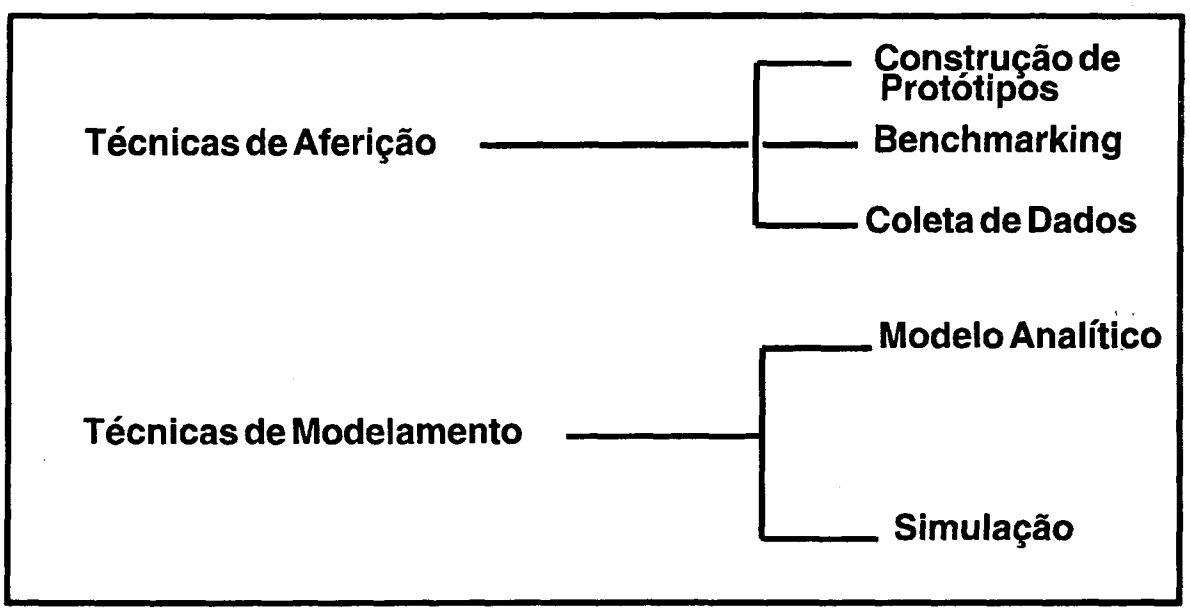

Figura 1.1 - Técnicas para Avaliação de Desempenho

Técnicas de Aferição:

Coleta de dados: Essa técnica deve ser conduzida sobre sistemas reais, operando em condições normais, através da inserção de algum hardware ou ao acréscimo de algum software, específicos para esse fim. Essa técnica é a que fornece resultados mais precisos, entretanto sua aplicação requer que o sistema em questão esteja fisicamente disponível. Cuidado especial deve ser tomado para que as próprias instruções que compõem essa técnica não interfiram nos resultados da análise [KO78, SA90a, FE92]. O tempo e o custo consumidos no emprego dessa técnica são determinados pelo tipo e complexidade da coleta de dados a ser efetuada.

Benchmarking: Em algumas situações, como na aquisição de novos equipamentos, é interessante que o desempenho de diferentes sistemas sejam comparados. Para isso, é necessário que o desempenho seja obtido com todas as máquinas operando sobre as mesmas condições, ou seja, a mesma carga de trabalho deve ser executada em cada uma delas. A carga de trabalho utilizada nesse procedimento (que pode ser uma aplicação qualquer do usuário ou um procedimento específico para este objetivo), é chamada "benchmark" [FE92]. Esse tipo de ferramenta, normalmente, não é de alto custo estando disponíveis no mercado

Construção de Protótipos: É uma técnica empregada quando se deseja obter dados de um sistema ainda não disponível. O protótipo nada mais é que uma aproximação simplificada do sistema real [FE92]. É um método que oferece boa precisão nos resultados, porém a construção de protótipos torna-se algumas vezes inviável devido aos custos envolvidos com a construção, testes e modificações do mesmo. Além disso essa técnica não é adequada para testar diferentes alternativas, devido ao custo e às dificuldades na modificação do sistema físico.

Técnicas de Modelagem:

Modelamento Analítico: Permite escrever uma relação funcional entre os parâmetros do sistema e os critérios de desempenho escolhidos, em termos de equações que podem ser resolvidas analiticamente [SO92]. Essa técnica fornece resultados precisos, mas a medida em que a complexidade do sistema sendo modelado aumenta, também aumenta a dificuldade da resolução analítica. Muitas vezes é possível a construção de modelos que representem bem o sistema e que possam ser resolvidos com algumas simplificações e suposições. Geralmente utilizam-se redes de filas para a representação do modelo e subseqüente solução [KO78,SA90,SA90a,SO92]. Essa técnica possui um custo relativamente baixo e o tempo para obter a solução desejada depende da complexidade do problema.

Simulação: Consiste em modelar o sistema para então transforma-lo em um programa que represente com fidelidade o sistema real. É uma técnica prognóstica que examina as conseqüências possíveis quando diferentes sequências de comportamentos do sistema são geradas [SA90a]. É uma técnica flexível podendo ser aplicada a sistemas existentes ou inexistentes. A dificuldade dessa técnica reside na validação dos resultados obtidos (essa mesma dificuldade é encontrada nos modelos analíticos) [KO78, MA87, SA90, SA90a, SO92].

Cada uma das abordagens pode ser útil dependendo do problema e do estágio do trabalho. A utilização de técnicas de aferição é viável quando o sistema está pronto e em uso ou existe um protótipo disponível. As desvantagens dessas técnicas envolvem situações nas quais o sistema não é existente e é necessária a realização de testes prognósticos sobre o sistema em desenvolvimento. Pode ser uma opção atrativa para a verificação do projeto final. Mas, ainda nesse caso, a obtenção dos dados requer cuidado. Por outro lado, técnicas de modelamento oferecem grande flexibilidade a baixo custo [SA90]. A adoção de tais técnicas é atrativa tanto quando o sistema é existente como quando não o é, e quando o uso das técnicas de aferição apresentam um custo elevado.

A decisão entre modelos analíticos e modelos de simulação deve ser baseada no contexto no qual serão utilizadas, não existindo uma regra geral para a escolha. Modelos analíticos oferecem em alguns casos soluções simplificadas, mas na maioria das vezes os sistemas são complexos determinando simplificações e suposições exaustivas para que as equações possam ser resolvidas. As ferramentas analíticas não são capazes de resolver uma gama de problemas práticos em diversas áreas.

Outra desvantagem do modelo analítico surge quando se deseja efetuar alguma modificação no sistema. Para efetivar a modificação deve-se refazer todo o modelo e ainda solucioná-lo. Os modelos de simulação permitem refletir essas alterações no modelo de uma forma mais simplificada. A simulação, entretanto, pode requerer um tempo extensivo de processamento para algumas aplicações.

Outra opção possível é adoção de um modelo híbrido [MA87], onde parte do problema é formulado pelas médias de um modelo analítico e a outra parte através da simulação. Isso auxilia a reduzir o tempo global de processamento da simulação.

Capítulo 2

Técnicas de Modelagem

O capítulo que segue tem por finalidade discutir conceitos básicos sobre técnicas de modelamento, estando organizado em três seções:

2.1 - Considerações Iniciais

Técnicas de modelamento para análise de desempenho de sistemas computacionais consistem na construção e análise de modelos representativos do sistema em estudo. Análise visa a obtenção de informações que auxiliem na resolução de algumas questões sobre o sistema real [MO86].

Um modelo é uma abstração do sistema em questão [MO86, FE92,SO92]. Durante a fase de modelamento deve-se ter como diretriz básica que somente as características essenciais do sistema devem ser refletidas no modelo. A inclusão de maiores detalhes pode introduzir complicações desnecessárias [SO92].

Observa-se que para cada meta ou proposta específica tem-se a construção de um modelo, que pode variar também segundo o modelador, pois, a abstração das características de um sistema é influenciada pela visão que se tem sobre o sistema e também pela experiência do modelador na modelagem de classes específicas de sistemas.

O número de detalhes ou características extraídos do sistema real determinam, na maioria dos casos, o grau de dificuldade para a solução do modelo. Existem modelos cujas soluções são extremamente triviais e outros cuja complexidade limita a aplicação dessa técnica. Em alguns casos, onde o modelo é complexo e sofisticado, deve-se submetê-lo a uma série de simplificações e suposições, até que seja viável a sua solução. Essas simplificações e suposições podem trazer prejuízos à análise de desempenho, uma vez que o sistema real pode ser destorcido no modelo. Quando não existem, entretanto, outras alternativas para a solução do modelo e tais técnicas são empregadas, os seguintes passos devem ser seguidos de modo a minimizar as consequências na análise dos resultados obtidos:

- Documentação de todas as simplificações e suposições efetuadas sobre o modelo. Dessa forma os resultados devem ser analisados assim que obtidos, na tentativa de se inferir as possíveis influências das técnicas utilizadas sobre os resultados.

- Deve-se ter consciência de que os resultados refletem apenas as tendências do comportamento do sistema real, e não é um diagnóstico exato do que realmente se passa pelo sistema.

Assim, a modelagem de um sistema pode ser vista como uma "simplificação" que descarta as características julgadas irrelevantes do sistema (figura 2.1) [MO86].

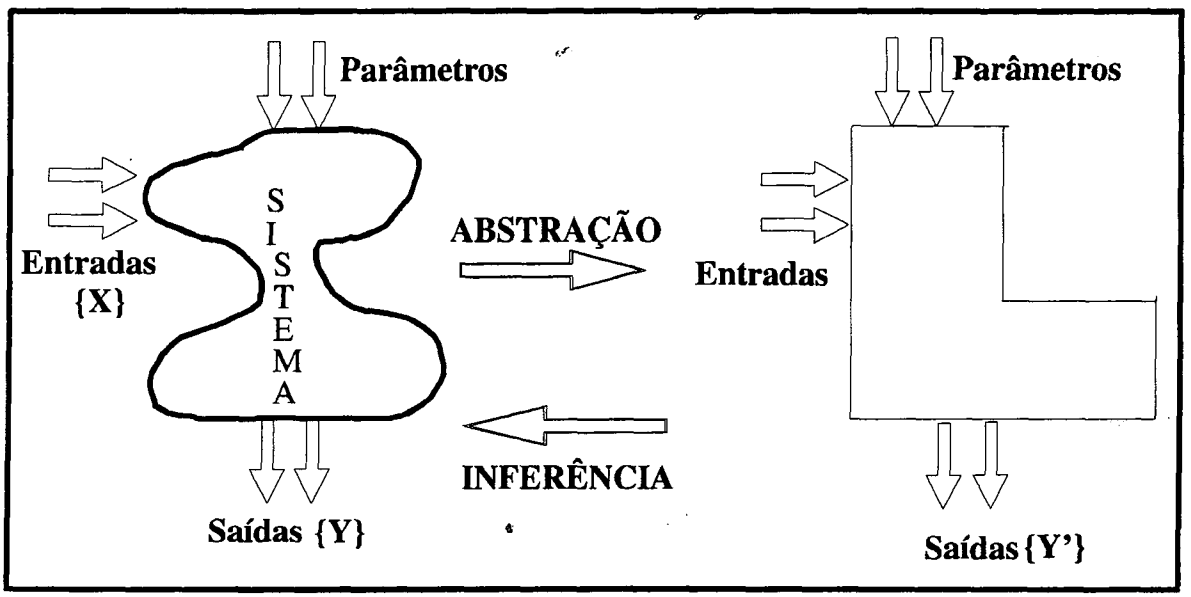

Figura 2.1 - Visão da Modelagem de um Sistema

Nota-se na figura 2.1 que deve haver uma estreita correspondência entre os dados de entrada e os parâmetros do sistema e do modelo. Os dados do sistema que estão sendo analisados devem ser inferidos a partir dos dados obtidos pelo modelo. Aqui a correspondência não é direta, devido a uma possível utilização de simplificações e suposições [MO86].

As maiores vantagens do emprego das técnicas de modelagem estão na flexibilidade - podem ser aplicadas a sistemas inexistentes e existentes; e na relação custo versus grau de precisão dos resultados obtidos [FE92]. Suas dificuldades estão na descrição das características essenciais do sistema e na validação do modelo, ou seja, a dificuldade aqui está em assegurar que o modelo é uma representação consistente do sistema real.

As principais ferramentas que compõem as técnicas de modelamento são o modelamento analítico e a simulação. O modelamento analítico será explorado nos próximos tópicos e o próximo capítulo será reservado à simulação.

2.2 Redes De Filas

A maioria das informações obtidas a partir de uma simulação estão relacionadas com os tempos de espera por serviços ou o número de entidades esperando por um tipo particular de serviço. As linhas de espera são denominadas filas, e modelagem de filas é uma parte essencial em todo o estudo de simulação [MA80].

Basicamente, uma rede de filas é constituída de entidades denominadas centros de serviço e um conjunto de entidades chamadas usuários (ou clientes), os quais recebem serviço nos centros. Um centro de serviço é constituído de um ou mais servidores (representando os recursos do sistema sendo modelado) que prestam serviço aos usuários, e uma área de espera (denominada fila) para usuários cujos pedidos excedem a capacidade do centro de serviço, e estão esperando pelo serviço requisitado.

Como exemplo de recurso que pode ser modelado como um centro de serviço, tem-se a unidade central de processamento (UCP) de um computador, no qual um cliente do sistema pode ser representado por uma tarefa ou "job". O cliente circula pelo sistema sendo modelado, requisitando serviço nos centros, e esperando em filas quando não existe servidor disponível.

Os centros de serviço podem ser classificados segundo o número de servidores no centro de serviço e o número de filas de espera para um dado servidor. Assim, pode-se ter centros com uma única fila (ou várias) e um servidor (ou vários), como discutido na próxima seção.

2.3 MODELOS BASEADOS EM REDES DE FILAS

Um modelo consiste em uma abstração do sistema a ser estudado. Assim, um sistema é modelado pela abstração de sua estrutura e características através de redes de filas.

O modelo mais simples é constituído de um centro de serviço com uma fila e um servidor [MA80, SO92]. O comportamento básico de muitos sistemas de filas pode ser representado pelo modelo da figura 2.2. Os clientes chegam a algum servidor (ou centro de serviço), e se o servidor está ocupado, o cliente une-se à fila associada com aquele servidor e espera por sua vez. Eventualmente, o cliente é selecionado para atendimento, de acordo com a disciplina da fila. O serviço requisitado é executado pelo servidor, e ao seu término o cliente deixa o sistema. Um exemplo desse tipo de modelo de filas consiste em um cinema com somente uma bilheteria, onde as pessoas se enfileiram para comprar seu ingresso.

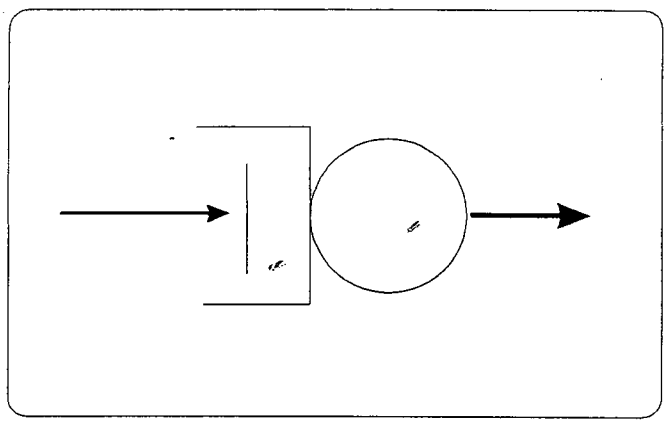

Figura 2.2 - Centro de Serviço: Uma Fila e Um Servidor

Uma variação do modelo básico contém um centro de serviço constituído de uma única fila que alimenta vários servidores (figura 2.3). Um exemplo bastante comum deste tipo de modelo consiste em uma agência bancária, onde os caixas atendem aos clientes que formam uma fila única.

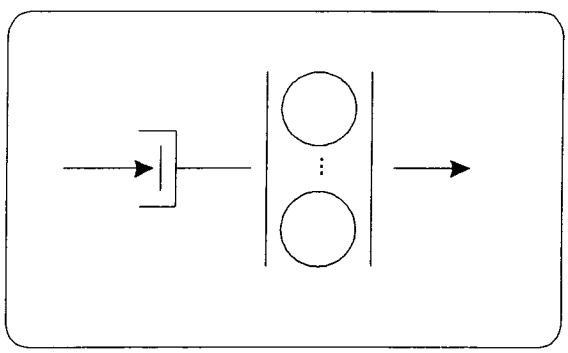

Figura 2.3 - Centro de Serviço: Uma Fila e Vários Servidores

Pode-se ter também modelos constituídos de centros de serviço formados por múltiplas filas associadas a um único servidor (figura 2.4). Pode-se considerar como exemplo um guichê de uma repartição pública que atende a usuários os quais formam duas filas diferentes. Uma das filas se destina aos aposentados, e a outra atende as demais pessoas. O atendente dá preferência a fila especial, só atendendo a outra fila quando não houver mais nenhum aposentado aguardando atendimento.

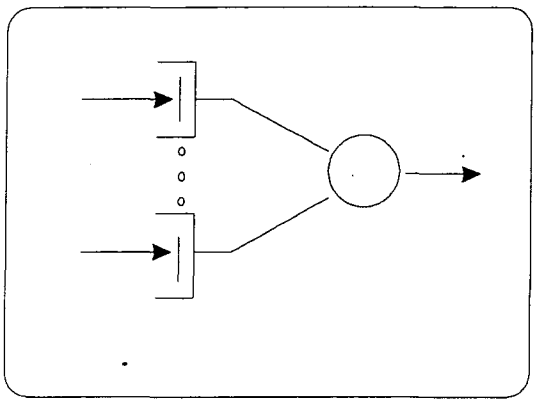

Figura 2.4 - Centro de Serviço: Múltiplas Filas e Um Servidor

O modelo mais complexo, constituído de múltiplas filas e múltiplos servidores é apresentado na figura 2.5. Como exemplo, pode-se considerar uma agência bancária onde os vários caixas atendem a clientes que formam duas filas: uma para os clientes em geral e outra para idosos, gestantes e portadores de deficiências físicas. Os clientes em geral só são atendidos quando não houver ninguém na fila especial.

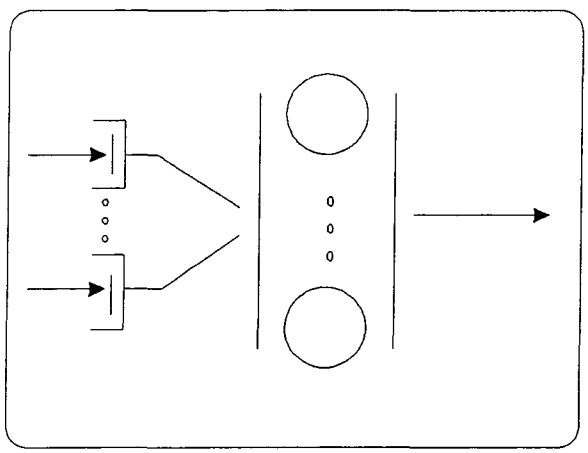

Figura 2.5 - Centro de Serviço: Múltiplas Filas e Múltiplos Servidores

Para cada um dos modelos descritos nesta seção, pode-se identificar alguns parâmetros necessários para a especificação completa do modelo. No primeiro modelo (uma fila e um servidor) deve-se definir qual a disciplina da fila, a taxa de chegada de novos usuários e o tempo necessário para executar o serviço.

O modelo constituído de uma fila e vários servidores necessita, além dos parâmetros vistos para o modelo mais simples, da seleção de um servidor quando mais de um está desocupado.

Para o terceiro modelo deve-se, além dos parâmetros necessários ao primeiro,

especificar a maneira pela qual uma fila é selecionada quando o servidor fica disponível (para retirar o próximo cliente), e como um cliente escolhe uma fila para aguardar pelo serviço.

O modelo mais complexo, constituído de centro de serviço com várias filas e vários servidores necessita de todos os parâmetros identificados para os modelos anteriores.

Assim, os seguintes parâmetros foram identificados nos modelos de redes de filas:

Tempo de Serviço

O tempo de serviço está relacionado com o tempo necessário para que um servidor execute uma tarefa. É usualmente especificado por uma distribuição de probabilidade, embora em alguns casos pode-se utilizar valores médios. A utilização de valores médios facilita a implementação mas acarreta problemas de precisão. Assim, na maioria dos casos são utilizados valores obtidos a partir de números aleatórios.

Algoritmos de Escalonamento (que regem a disciplina das filas de espera)

A disputa por recursos do sistema geralmente leva à formação de filas de espera, onde os clientes aguardam sua vez de serem atendidos. Existem vários algoritmos, denominados algoritmos de escalonamento, usados para decidir qual usuário deve entrar em serviço quando um servidor se encontra disponível (tais algoritmos são freqüentemente conhecidos como disciplina da fila). Assim uma disciplina de fila é uma regra para adicionar e remover clientes de uma linha de espera. Diferentes tipos de algoritmos de escalonamento podem ser aplicados em um estudo de simulação [MA80, SO92], dentre eles cabe ressaltar:

- **FCFS("first come first served"):** consiste em uma fila de espera padrão, a ordem de chegada é a ordem para o atendimento no centro de serviço;

- **LCFS("last come first served"):** essa disciplina tem a estrutura de uma pilha, onde o último usuário a chegar ao centro é o que entrará em serviço assim que houver disponibilidade do servidor (por exemplo carga a ser colocada em um caminhão);

- **RR ("round robin"):** cada cliente é atendido por um pequeno intervalo de tempo, denominado "quantum". Caso não tenha sido suficiente, o cliente volta para o final da fila de espera, e o cliente seguinte entra para receber seu "quantum" de serviço. O processo se repete até que o serviço requisitado pelo cliente se complete;

- **Tempo de serviço:** o usuário com o maior (menor) tempo de serviço esperado é retirado da fila. O problema com esse algoritmo é que a fila deve ser ordenada pelo tempo de serviço;

- **Uso de prioridades:** as formas de escalonamento com prioridade são utilizadas para dar preferência a alguns clientes, podendo ser com ou sem preempção. Quando não existe preempção, um cliente em atendimento não é afetado quando um de mais alta prioridade chega, enquanto que no caso de existir preempção a chegada de um cliente com prioridade maior do que o que está sendo atendido faz com que este seja retirado e colocado na fila. O cliente de mais alta prioridade é atendido, e o que foi retirado só retorna ao servidor quando todos aqueles de prioridade maior já tiverem sido atendidos.

Uma importante consideração a ser feita é com relação ao tamanho da fila de espera, que pode ter comprimento limitado ou ser infinita, isto é, não fazer restrições quanto ao número de clientes que aguardam por um serviço.

Tempo Entre-Chegadas

É necessário alguma função que produza o tempo da próxima chegada ao sistema. No caso mais simples, essa função pode ter valores constantes, porém este caso não irá representar o modelo de modo realista. A função que descreve chegadas de clientes ao sistema deve produzir valores aleatórios que correspondam aos tempos entre-chegadas no sistema real [MA80].

Algoritmos de Escolha de Servidores

Quando um usuário requisita serviço e mais de um servidor do centro de serviços está disponível, deve-se escolher qual deles irá atender ao pedido. Esse algoritmo de seleção é muito similar a uma disciplina de fila, podendo ser o servidor selecionado aleatoriamente, o que está a mais (menos) tempo desocupado, ou ainda segundo algum tipo de prioridade [MA80].

Algoritmos para Escolha de Qual Fila Retirar o Próximo Cliente a ser Atendido

Neste caso, quando o servidor se torna disponível, deve-se determinar de qual fila retirar o próximo usuário. A escolha pode ser feita seguindo algum tipo de prioridade, ou então aleatoriamente.

Algoritmos para o Cliente Escolher em Qual Fila irá Entrar

Um cliente, quando chega ao centro de serviço, deve optar por uma fila de espera, à sua livre escolha ou então com alguma forma de prioridade.

Capítulo 3

Simulação

3.1 Desenvolvimento do Modelo e Testes

Modelos de um sistema podem ser classificados como modelos de mudança discreta e modelos de mudança contínua, de acordo com o modo no qual o modelo representa as mudanças de estado no sistema em estudo [MA75, MA80, SO92, PE91]. Os sistemas podem ser discretos ou contínuos.

- Sistemas Discretos: nesse tipo de sistema as mudanças de estado ocorrem em pontos discretos do tempo. Uma simulação discreta pode ser vista como uma descrição dos eventos que ocorrem a cada intervalo de tempo;
- Sistemas Contínuos: nesses sistemas as mudanças de estado ocorrem continuamente no tempo. Como exemplo pode-se citar modelos para analisar dissipação de calor.

O passo chave no processo de modelagem, segundo MacDougall [MA87], consiste em abstrair as características de um sistema em um modelo, valendo toda a experiência do modelador com trabalhos desse gênero. O processo de modelagem pode ser dividido em três fases: desenvolvimento, teste e análise, com cada fase compreendendo vários passos, como mostra a figura 3.1. Ressalta-se que embora o processo de desenvolvimento seja descrito de forma seqüencial, ele de fato não é, pois normalmente existe interação entre os diversos passos. É importante destacar também que alguns dos passos que compõem o desenvolvimento de uma simulação são comuns a uma resolução analítica.

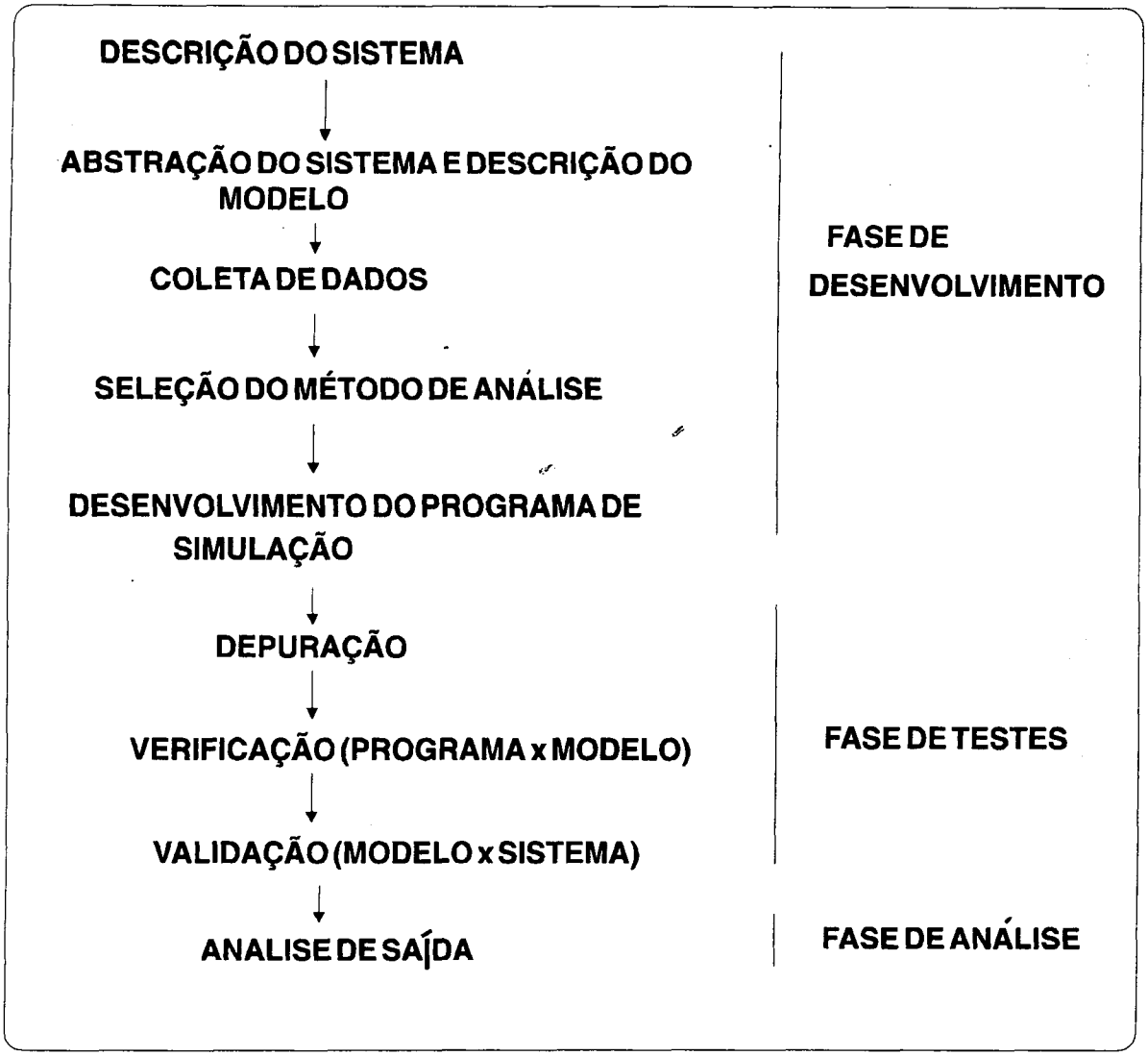

Figura 3.1 - O Processo de Modelagem e Análise

3.1.1 Fase de Desenvolvimento

Descrição do Sistema

O primeiro passo na elaboração de uma simulação é o entendimento do sistema considerado. O sistema deve ser definido com grande precisão e as metas pretendidas devem ser claras e objetivas [MA80, PE91]. MacDougall, em [MA87], destaca que o tempo gasto em organizar e escrever essa descrição é compensado devido aos possíveis erros eliminados. A forma da descrição depende do tipo de sistema sendo modelado, e uma das maneiras de representar o sistema é através de um diagrama de fluxo.

Abstração do Sistema e Descrição do Modelo

Uma vez formalizado o problema e definido os objetivos a estudar, inicia-se o processo de construção do modelo. Um modelo é uma abstração de um sistema, e representa uma visão particular do mesmo. Ao criar o modelo, uma tarefa relativamente difícil é a decisão sobre quais elementos do sistema devem ser incluídos no modelo, e a quantidade de detalhes incluída deve ser baseada no propósito para o qual o modelo está sendo construído [MA87, PE91, SO92]. Nesse passo descreve-se o que deve ser representado no modelo, e a escolha do método (modelos são descritos em termos do método utilizado para obter medidas de desempenho, podendo ser modelo analítico ou de simulação) fica para o passo seguinte.

Coleta de Dados

A fase de formulação do modelo gera os requisitos para os dados de entrada, que servirão como parâmetros do mesmo. Quando a modelagem é feita sobre um sistema existente, os parâmetros para o modelo podem ser medidos diretamente (via instrumentação de "hardware" ou "software") ou indiretamente [MA87].

Seleção do Método de Análise

O passo seguinte, seleção do método de análise, consiste em escolher entre o método analítico ou simulação. Diversos autores [MA87, DA89, SO92] defendem o uso de ferramentas analíticas sempre que for viável. Porém, em diversos casos a utilização de soluções analíticas não é adequada, principalmente devido a falta de flexibilidade (uma simples modificação no projeto pode implicar em reformulação do modelo) e a dificuldade em representar muitos detalhes (aumentar os detalhes em um modelo analítico pode aumentar a complexidade da solução analítica). Nesses casos, simulação deve ser considerada.

Desenvolvimento do Programa de Simulação

O próximo passo, no caso de escolha de modelagem por simulação, consiste em traduzir o modelo para uma forma aceitável pelo computador. A seleção do computador, linguagem e ferramentas depende do sistema sendo simulado e dos recursos disponíveis, sendo que pode-se seguir os seguintes enfoques [SA90a, SA90b]:

- **desenvolvimento do programa de simulação em uma linguagem convencional (como C, Modula 2, Pascal):** neste caso, o programador é responsável por criar todo o ambiente necessário para a simulação. Oferece a vantagem de que o programador (ou modelador) tem liberdade para escolher uma linguagem que seja de seu conhecimento, porém tem que desenvolver todas as estruturas exigidas num ambiente de simulação;

- **utilização de um pacote de uso específico para criar o programa de**

simulação: deve existir um pacote que se ajuste às necessidades do sistema sendo modelado. Nesse caso, fica fácil construir o programa, já que normalmente só é necessário parametrizar o modelo pronto. Essa alternativa representa para o programador (modelador) a aprendizagem de uma nova linguagem, além de fornecer pouca flexibilidade a mudanças, por ser muito específico para um determinado tipo de sistema;

- **uso de linguagens de simulação de uso geral:** permitem que um número maior de sistemas seja modelado do que em pacotes específicos. Nesse caso, as linguagens já contém todas as estruturas necessárias para a criação de um ambiente de simulação, livrando o programador da necessidade de executar tal tarefa. Como estas linguagens são menos específicas, permitem a simulação de um número maior de sistemas. O usuário pode ter que aprender uma nova linguagem;

- **através de uma extensão funcional de uma linguagem de programação de uso geral:** utilizando este enfoque, tem-se os aspectos gerais de uma linguagem de programação unidos ao caráter específico de uma linguagem de simulação. Isso permite ao usuário codificar seu programa utilizando primitivas de uma biblioteca e uma linguagem convencional. Pode-se ter extensões utilizando diferentes linguagens como por exemplo C (SMPL [MA87], EFC [SO92a]) e Modula 2 (EFM2 [SP92]).

3.1.2 Fase de Testes

Depuração do Programa

O programa de simulação deve ser conferido para garantir que está livre de erros (de programação, de lógica, etc), devendo produzir resultados que pareçam coerentes.

Verificação

O passo seguinte no processo de modelagem é verificar se o programa de simulação é uma implementação válida do modelo. Para modelos pequenos, pode ser feito por inspeção, para modelos maiores análise mais detalhada se torna necessária. A técnica normalmente utilizada consiste em uma revisão cuidadosa no programa e no modelo [SA90b]. A verificação pode ser feita através da comparação com modelos analíticos. Provavelmente existirão variações nos resultados, porém deve-se levar em conta a variabilidade estatística dos resultados, referentes à geração de amostras aleatórias. A utilização de diferentes sementes de números aleatórios em diversas execuções da simulação pode minimizar esta variabilidade [MA87].

Validação

O modelo de simulação deve ser uma representação válida do sistema atual, ou

seja, deve reproduzir o comportamento do sistema com fidelidade, para satisfazer aos objetivos da análise. É tarefa da validação comparar o modelo com o sistema, e se o resultado não for satisfatório, a lógica do modelo, bem como as suposições e simplificações efetuadas e os dados coletados devem ser analisados cuidadosamente [DA89]. Quando o sistema sendo modelado já existe, medidas do mesmo são comparadas com resultados de uma simulação do sistema e se os resultados coincidem, o programa de simulação é considerado válido, podendo ser utilizado para análise do sistema e para predizer os efeitos de modificações no mesmo [SA90b]. Quando a simulação é feita sobre um projeto, a validação é feita examinando-se o projeto do sistema, e cada suposição e abstração é justificada. A validação pode também se basear em comparações com outros modelos do sistema já validados, como por exemplo modelos analíticos [SO92, SA84].

Técnicas para verificação e validação são demoradas mas importantes quando se deseja que o modelo fique livre de erros e tenha credibilidade, e consistem em um dos aspectos mais críticos de um estudo via simulação [MA80, SA84].

3.1.3 Fase de Análise

Análise de Saída da Simulação

Uma questão importante em simulação é por quanto tempo (e quantas vezes) executar o programa de simulação. As entradas da simulação são obtidas através da geração de variáveis aleatórias de distribuições de probabilidades, as saídas da simulação são funções dessas variáveis, e portanto a análise de saída é um problema estatístico (os resultados das execuções de uma simulação devem ser cuidadosamente analisados para entender o seu significado). Existem vários métodos descritos na literatura para determinar quando parar um programa de simulação, baseado no cálculo de um intervalo de confiança para a média de uma variável de saída . Ressalta-se que métodos para análise de saída auxiliam na obtenção de uma estimativa precisa de uma medida de desempenho de um modelo; o quão precisamente tal medida representa o desempenho do sistema depende da validade das suposições utilizadas para construir o modelo [MA87].

3.2 Simulação Discreta

Um modelo de simulação descreve o comportamento dinâmico de um sistema, isto é, a maneira como este executa um determinado trabalho. A sensibilidade dos resultados da simulação à mudanças nos parâmetros de entrada pode ser avaliada pela realização de uma série de repetições da simulação, variando-se tais parâmetros. Esse procedimento permite analisar alternativas para o sistema modelado.

Os objetos de um sistema são definidos como sendo entidades [SO92]. O objetivo de um modelo para simulação é reproduzir as atividades das entidades engajadas e a partir daí conhecer algo sobre o comportamento e desempenho do

sistema.

Um sistema pode ser visto como uma coleção de processos interativos, cada processo composto por diversas atividades e com as interações controladas e coordenadas pela ocorrência de eventos [MA87]. A relação entre esses conceitos pode ser melhor entendida pela figura 3.2.

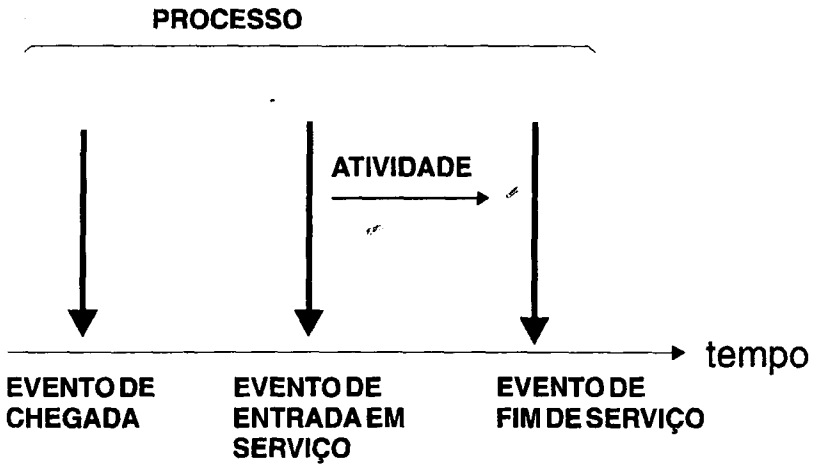

Figura 3.2 - Relação entre Evento, Processo e Atividade

- **atividade:** é a menor unidade de trabalho, e depende da visão particular do sistema (toda atividade tem um tempo associado). Uma atividade pode ser por exemplo a execução de um passo de uma instrução ou uma tarefa toda, dependendo do nível de detalhamento considerado;

- **processo:** um processo é um conjunto de atividades logicamente relacionadas, como por exemplo a execução de uma operação de disco (composta das seguintes atividades: posicionamento, atraso de rotação, transferência dos dados). A definição dos processos de uma simulação depende do nível de abstração adotado, por exemplo um programa inteiro pode ser visto como um processo (compreendendo atividades de computação, leitura e escrita de dados). Em um outro nível, a execução de uma atividade de escrita pode ser vista como um processo compreendendo posicionamento, latência e transferência de dados. Tal programa poderia ser visto ainda como uma atividade de um sistema maior. Assim, como ilustra a figura 3.3, uma operação definida como uma atividade em um nível de abstração pode ser definida como um processo compreendendo um conjunto de atividades em um outro nível de abstração, e a distinção entre atividade e processo depende do nível de detalhamento;

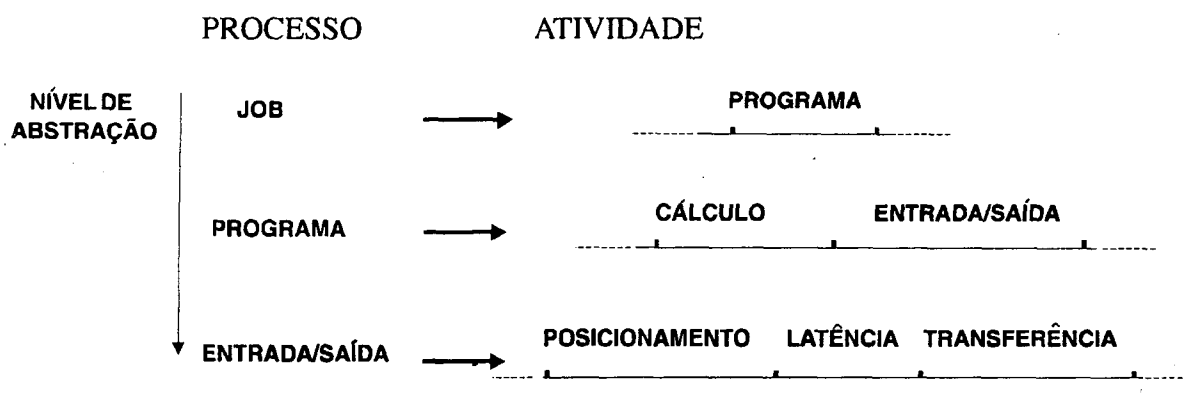

Figura 3.3 - Níveis de Abstração de Processo / Atividade

- **eventos:** um evento consiste em uma mudança instantânea do estado de alguma entidade do sistema (um evento ocorre em um ponto isolado do tempo, no qual decisões devem ser tomadas de forma a iniciar ou terminar uma atividade). A simulação do sistema é produzida pela execução da lógica associada a cada evento, em seqüência ordenada no tempo [SO92].

3.3 Modelagem para Simulação Discreta

Um modelo para simulação discreta pode ser organizado baseado em três entidades básicas, resultando em uma das seguintes orientações:

- **orientação a evento:** pela definição das mudanças nos estados que podem ocorrer em cada tempo de evento;
- **orientação a atividade:** através da descrição das atividades nas quais as entidades do sistema se envolvem;
- **orientação a processo:** por meio da descrição dos processos através dos quais as entidades fluem.

A seguir é apresentada uma breve descrição dos ambientes de simulação voltados para cada uma das orientações.

3.3.1 Simulação Orientada a Evento

Na simulação orientada a evento, um sistema é modelado pela definição das mudanças que ocorrem no tempo de evento. O modelador deve determinar os eventos que podem causar a mudança no estado do sistema e desenvolver um algoritmo associado a cada evento. O sistema é simulado pela execução dos algoritmos em uma seqüência ordenada no tempo.

Essa seqüência ordenada no tempo é mantida em uma estrutura de lista, denominada lista de eventos futuros, na qual cada entrada contém o tempo de ocorrência de um evento juntamente com um identificador que define as ações que devem ser consideradas quando o evento ocorrer. O identificador de evento correspondente é inserido na lista de eventos futuros por um mecanismo de escalonamento. Após completar todas as operações possíveis em um particular tempo de simulação, o mecanismo escalonador é ativado para selecionar o próximo evento a ser executado [MA75]. Escalonar um evento significa então gerar o tempo em que ocorrerá o próximo evento e inseri-lo na lista de eventos futuros, em forma ordenada pelo tempo de ocorrência.

Como um exemplo, considera-se o atendimento a clientes, por caixas de um banco [MA80]. Ao chegar ao banco, o cliente entra na fila e espera sua vez de ser atendido. Após terminar suas operações no caixa, o cliente deixa a agência bancária. Mudanças de estado no sistema ocorrem devido aos seguintes eventos:

- um cliente chega para pagar suas contas;
- um cliente já foi atendido por um caixa e parte do sistema.

A lógica associada aos eventos pode ser descrita da seguinte forma [SO92]:

Evento Chegada de Cliente

```
Escalone a próxima chegada
Se o caixa está ocupado então
  Aumente o número de clientes na fila
Senão
  Torne o estado do caixa ocupado
  Escalone o fim de atendimento para tempo_corrente +tempo_atendimento
```

Evento Término de Atendimento

```
Se existe algum cliente na fila então
  Retire um cliente da fila para atendimento
  Escalone o fim de atendimento para tempo_corrente +tempo_atendimento
Senão
  Torne o estado do caixa desocupado
```

O programa de simulação pode ser organizado da seguinte forma:

```
Escalone a primeira chegada
Selecione o primeiro evento
Avance o tempo de simulação para o tempo de ocorrência do evento
Se final de simulação então
  Imprima estatísticas
Senão
  Se evento = Chegada de Cliente então
    Trata chegada de Cliente
  Senão
    Trata término de atendimento
```

3.3.2 Simulação Orientada a Processo

Em simulação orientada a processo, o sistema sendo simulado é visto como uma coleção de processos interativos, controlados e coordenados pela ocorrência de eventos [MA75]. Como visto na seção 3.2, um processo é um conjunto de atividades logicamente relacionadas.

O programa de simulação é organizado como um conjunto de processos, os quais executam concorrentemente durante a simulação. A descrição de um processo tem a forma de um procedimento de uma linguagem de programação [MA75].

Como na orientação a eventos, também existe um mecanismo de escalonamento e uma estrutura de lista, com a diferença de que cada entrada na lista define um processo e seu ponto de reativação (para processos que são bloqueados e novamente postos para executar).

Basicamente, uma simulação orientada a processo deve seguir os seguintes passos:

- definir as entidades do sistema;
- criar um processo para cada entidade descrevendo suas etapas;
- executar concorrentemente os processos.

No exemplo dos clientes sendo atendidos por caixas de uma agência bancária, descrito na seção anterior, seguindo este tipo de orientação temos o processo cliente organizado da seguinte maneira:

Processo Cliente

Se o caixa está desocupado então
  Torne o estado do caixa ocupado
  O cliente ocupa o caixa pelo tempo de atendimento (escalone processo para tempo_corrente + tempo_atendimento)
  Torne o estado do caixa desocupado
  O cliente parte do sistema
Senão
  O cliente entra na fila de espera

A chegada de novos clientes ao sistema (o banco) pode ser descrita como no processo Chegada de Clientes:

Processo Chegada de Clientes

Indique chegada de novo cliente
Escalone processo para tempo_corrente + tempo_entre_chegadas

O programa de simulação executa concorrentemente vários processos Cliente, que chegam segundo uma distribuição de probabilidade, requisitam serviços e vão embora.

3.3.3 Simulação Orientada a Atividade

Utilizando essa orientação, o modelador descreve cada atividade possível para cada entidade do sistema, definindo as condições que causam seu início e seu término. Os eventos que iniciam ou finalizam uma atividade não são escalonados pelo modelador, mas são iniciados a partir das condições especificadas para a atividade. Conforme o tempo de simulação avança, as condições para o início ou término de uma atividade são examinadas, e no caso de alguma condição ser satisfeita, a atividade correspondente é escalonada para execução no tempo de simulação corrente. Esta orientação é pouco utilizada para a modelagem de sistemas [SO92, MA75, MA80].

No exemplo dos clientes sendo atendidos por caixas de um banco, temos as seguintes atividades:

- chegada de um cliente;
- término de atendimento.

A estrutura básica do programa de simulação é mostrada a seguir.

```
Início
  tempo de simulação  = 1
  Enquanto não fim de simulação faça
    Verificar o conjunto de atividades
    Se chegada de cliente então
      Processa chegada
    Se término de atendimento então
      Processa término
    tempo de simulação =tempo de simulação + 1
  Imprima Estatísticas
```

Para a implementação de uma simulação segundo uma das orientações apresentadas, existem linguagens e pacotes de simulação que possibilitam ao modelador escrever programas de simulação orientados a evento, processo ou atividade, como abordado na seção 3.4.

3.4 Implementação de um Modelo de Simulação

Como visto na seção 3.1.1, pode-se usar os seguintes enfoques para a implementação de um modelo de simulação:

- linguagens de programação convencionais;
- linguagens de simulação;
- pacotes de uso específico;
- extensões funcionais.

Um requisito importante para as linguagens de simulação é a presença de mecanismos para representação do tempo. Enquanto o sistema sendo modelado executa em tempo real, a simulação trabalha com um relógio próprio, que marca a passagem do tempo no programa de simulação (para refletir a passagem desse tempo, pode ser utilizada uma variável, atualizada a cada avanço do relógio simulado) [MA75]. Outro requisito consiste em oferecer facilidades para coleta de estatísticas e emissão de relatórios.

O uso generalizado de simulação como uma ferramenta de análise de sistemas deu origem a uma série de linguagens especificamente projetadas para este fim. Soares [SO92] classifica os pacotes para simulação existentes em pacotes de uso específico e linguagens de simulação.

3.4.1 Linguagens de Simulação

Essas linguagens de simulação são projetadas para a modelagem de sistemas

de vários tipos [SO92].

As linguagens para simulação discreta são classificadas em orientadas a atividades, processos ou eventos, baseado na organização imposta [MA87]. Cada classe de linguagem impõe ao modelador uma visão diferente do sistema, o que implica no fato do modelo de simulação poder ser visto como um conjunto de atividades, de processos ou de eventos.

3.4.1.1 Linguagens Orientadas a Atividade

Nesse caso, o programa de simulação é organizado como um conjunto de descrições de atividades, uma para cada tipo de atividade no sistema.

Soares, em [SO92], afirma que para certos tipos de problemas o uso dessa abordagem provê um modelo conciso, adequado para situações onde a duração da atividade é indefinida. A organização do modelo de simulação por atividades não tem sido largamente adotada [MA75].

Um exemplo de linguagem de simulação orientada a atividade é CSL [MA75].

3.4.1.2 Linguagens Orientadas a Evento

Nesse tipo de orientação o programa de simulação é organizado como um conjunto de rotinas ou seções de eventos [MA75]. A execução das rotinas de evento é sequencializada por um mecanismo de lista (chamada lista de eventos futuros).

As linguagens orientadas a evento tendem a impor uma visão global e de alto nível do sistema ao modelador. Existe ainda a tendência de se englobar operações sobre atividades logicamente não relacionadas em um único evento. Para sistemas complexos, isto pode ocasionar a perda de identidade com a estrutura do sistema real, devido a extensão do programa de simulação, e a programas mais difíceis de serem modificados [MA75].

Devido aos fatores apresentados, linguagens de simulação orientadas a evento são adequadas a modelos de pequeno e médio porte. Com o aumento da complexidade, MacDougall apresenta o uso de linguagens orientadas a processo como melhor alternativa.

Exemplos de linguagens de simulação orientadas a evento são SIMSCRIPT [KO78, MA75], SLAM II [SO92].

3.4.1.3 Linguagens Orientadas a Processo

Saydam [SA85] justifica o uso de linguagens orientadas a processo em análise de sistemas que vem se tornando cada vez mais complexos, como ocorre em áreas como a robótica, projeto de sistemas computacionais e redes de comunicação. Segundo o autor, linguagens orientadas a processo são a ferramenta adequada para a

modelagem desse tipo de sistema que necessitam de altos investimentos e portanto exigem que o comportamento dos parâmetros custo-projeto-desempenho sejam investigados antes da construção do sistema.

O sistema sendo modelado é visto como uma coleção de processos interativos. Um processo tem as seguintes características [SA85]:

.pode ser ativado, ficar em estado suspenso ou encerrar sua execução em certo instante de tempo ou baseado em alguma condição;
.uma vez ativado, repete o seu comportamento até ser colocado no estado suspenso ou terminar sua execução.

As linguagens orientadas a processo apresentam a vantagem de permitir uma representação mais natural do modelo. Essa característica é importante em sistemas de grande porte, pois o programa de simulação tende a se tornar uma descrição do comportamento do sistema real, estabelecendo assim maior identidade com o mesmo (o que facilita a tarefa de garantir que o modelo é uma representação válida do sistema) [MA75].

Um exemplo de linguagem orientada a processo que vem sendo muito utilizada é GPSS (General Purpose Simulation System) [SO92]. Um modelo GPSS é construído pela combinação de um conjunto de blocos padrão em um diagrama de blocos que define a estrutura lógica do sistema.

Outros exemplos de linguagens de simulação orientadas a processo são Q-GERT [SO92], ASPOL [SO92, MA75], SIMULA [MA75, SA85], SOL [MA75], SIMPL/I X [SA85], RESQ, SLAM II [SO92].

3.4.2 Pacotes de Simulação de Uso Específico

Os pacotes de uso específico são voltados para a avaliação de sistemas particulares. Devido ao fato de serem muito específicos para uma dada aplicação, estes pacotes oferecem pouca flexibilidade a mudanças.

A formulação do modelo é construída na própria ferramenta, e os parâmetros do modelo são especificados através de uma linguagem relacionada com o sistema modelado. Alguns exemplos são [SO92]:

- BEST/1, CMF, VMPredictor: apropriados para modelagem de sistemas computacionais e respectivos sistemas operacionais;
- PET, NETWORK II.5: utilizados na modelagem de redes de computadores;
- IDSS 2.0: aplicados na modelagem de sistemas aeroespaciais;
- CROPS: usado em modelagem de sistemas agrícolas.

3.4.3 Linguagens de Programação

Em princípio, toda linguagem de programação é uma candidata a linguagem de simulação [SA85, PE91]. Porém, linguagens como Fortran, C, Pascal, Modula-2 são projetadas para uso geral, não fornecendo todas as ferramentas (estruturas de dados, facilidades para manipulação de listas e abstração de dados) necessárias para um ambiente de simulação. Procedimentos específicos tornam-se necessários para criar e manipular as interações dessas estruturas de programação.

Para desenvolver um programa de simulação em uma linguagem convencional (como Modula 2, Pascal, C) o programador deve criar todo um ambiente necessário para a simulação. Isto pode ser uma desvantagem pois exige conhecimentos na área de simulação e programação.

Uma vantagem de se implementar um programa de simulação em uma linguagem de programação é que o programador não necessita aprender uma nova linguagem, pois pode trabalhar com a que está habituado. Porém o programador pode não possuir o conhecimento necessário para criar todo o ambiente necessário à simulação, então uma solução pode ser o uso de extensões funcionais juntamente com uma linguagem que seja dominada pelo mesmo. Extensões funcionais são tratadas a seguir.

Destaca-se que o programador pode desenvolver um ambiente orientado a eventos, processos ou atividades, dependendo da aplicação.

3.4.4 Extensões Funcionais

Extensões são bibliotecas que quando inseridas em linguagens hospedeiras (como C ou Modula 2) compõem um ambiente completo de simulação. Essas extensões facilitam o trabalho do programador, que não necessita aprender uma nova linguagem.

Alguns exemplos são:

- extensões da linguagem C: SMPL [MA87] (orientada a evento), CSIM [ED92] (orientada a processo) e EFC [SO92a] (orientada a processo);
- extensões da linguagem Modula 2: EFM2 [SP92], HPSIM [SH88] (orientadas a processo).

SMPL é uma extensão da linguagem C. Na visão SMPL de sistemas, existem três tipos de entidades: facilidades, fichas e eventos.

- facilidade: representa algum recurso do sistema sendo modelado;
- fichas: representam as entidades ativas do sistema, o comportamento dinâmico do sistema é modelado pelo movimento de fichas através de um conjunto de facilidades. Pode ser por exemplo uma tarefa em um modelo de um sistema

computacional, ou um pacote em um modelo de uma rede de computadores;
- evento: representa uma mudança de estado de qualquer entidade do sistema.

3.4.5 Escolha de uma Linguagem para Simulação

O uso de uma linguagem adequada para implementar a simulação torna-se importante, uma vez que a construção do programa de simulação é freqüentemente uma fase bastante complexa do processo de simulação [KO78]. Muitas vezes a seleção de uma linguagem para simulação é realizada muito mais pela disponibilidade da linguagem, do que por suas características estruturais.

Ao utilizar uma linguagem de simulação, o enfoque (evento, processo ou atividade) já vem implícito na mesma, o mesmo ocorrendo com as extensões. Porém, ao se utilizar uma linguagem de programação, o enfoque (evento, processo, ou atividade) é de responsabilidade do programador.

MacDougall [MA75] aconselha a escolha de linguagens orientadas a processo para modelagem de sistemas de grande porte, devido à maior identidade com o sistema real. O uso de linguagens orientadas a evento é mais adequado para modelar sistemas de médio e pequeno porte. Linguagens orientadas a atividade são ineficientes, merecendo pouco destaque na literatura. Esses conceitos são válidos para as linguagens de programação, quando o programador cria o ambiente para simulação, e para as extensões.

Vários fatores devem ser analisados na escolha de uma linguagem de simulação [SO92]:

- portabilidade: disponibilidade da linguagem em outros computadores, incluindo microcomputadores e supercomputadores;
- tempo de execução: velocidade de compilação e execução;
- capacidade de memória: gerenciamento adequado do espaço de armazenamento;
- depuração: facilidade de depuração, documentação adequada;
- aprendizado: facilidade de aprendizado da linguagem e treinamento disponível;
- considerações de processamento: disponibilidade de facilidades para coleta estatística e para processamento de listas. Geração de relatórios padrão e facilidade para geração de relatórios personalizados.

3.5 Conclusão

Simulação é um método que tenta imitar o comportamento do sistema em estudo, constituindo-se em um experimento estatístico que observa o comportamento do modelo no decorrer do tempo. É uma técnica que tem sido amplamente utilizada nas mais diversas áreas do conhecimento humano, devido à sua flexibilidade e baixo custo. Tais características também a tornam preferível em situações onde outras técnicas (como modelagem analítica e instrumentação) não apresentam solução ou então não podem ser aplicadas.

Desenvolver um modelo de simulação envolve seguir várias fases, compreendendo o desenvolvimento, teste e análise dos resultados obtidos. Na fase de desenvolvimento define-se um modelo para o sistema a ser simulado, identificam-se e obtém-se os dados que vão alimentar o programa de simulação (considerações sobre os parâmetros do modelo de simulação são feitas no capítulo 4). O programa pode ser desenvolvido utilizando-se uma linguagem de programação, uma linguagem de simulação, um pacote de uso específico ou então uma extensão funcional. Na fase de testes faz-se a depuração do programa, que é verificado e validado para que o modelo seja uma representação razoável do sistema real. Na fase de análise os métodos estatísticos servem de apoio a tomadas de decisão como com relação ao número (e tamanho) das execuções de um programa de simulação, para que se obtenha resultados razoáveis.

Modelos de simulação para sistemas discretos usualmente envolvem disputa por recursos, levando a formação de filas onde os componentes do sistema devem esperar até que os recursos estejam disponíveis. Os atrasos entre as mudanças de estado são determinados estatisticamente, com o intervalo exato selecionado de acordo com alguma amostragem de uma distribuição de probabilidade [SH88]. Outros métodos estatísticos, como por exemplo para determinar o número de execuções de um programa de simulação, chamada de análise de saída [MA87], são também necessários.

Capítulo 4

Técnicas de aferição

Este capítulo discute a utilização, vantagens e desvantagens das principais técnicas de aferição.

4.1 - Considerações Iniciais

Técnicas de aferição são ferramentas utilizadas para analisar o desempenho de um sistema através de sua experimentação. Essa experimentação pode ser efetuada utilizando software (como por exemplo os "benchmarks") ou hardware (como por exemplo monitores de hardware).

Essa abordagem fornece resultados mais precisos que as técnicas de modelamento, mas sua aplicação requer a disponibilidade do sistema ou de seu projeto em fase final de desenvolvimento. Outra dificuldade encontrada é como experimentar o sistema de modo que a própria experimentação não influencie o comportamento do sistema e portanto não interfira na análise dos resultados. Outro problema também ocorre quando há necessidade de se verificar diferentes alternativas de sistemas; essa tarefa implica em altos custos para as alterações e testes, além da própria dificuldade de executá-la. [LU71, SA90, SA90a].

Entre as principais técnicas de aferição citam-se: a construção de protótipos, "benchmarks" e coleta de dados. Essas abordagens são apresentadas no diagrama da figura 4.1, sendo discutidas nos próximos tópicos deste capítulo.

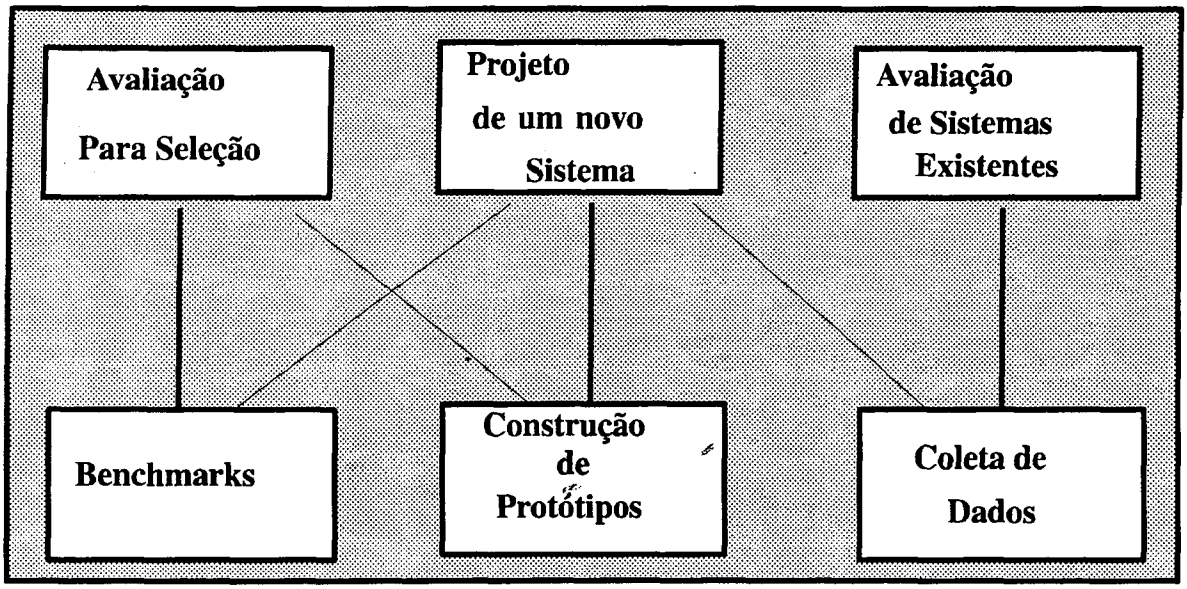

Figura 4.1 - Propostas de Avaliação e as Técnicas de Aferição

4.2 - Construção de Protótipos

Um protótipo é uma versão funcional de um produto, mas na qual faltam certas características não essenciais, que estarão presentes somente no sistema real [ST87].

Existem duas alternativas para a análise de desempenho de um novo sistema que está em fase de projeto. A primeira alternativa consiste na construção de todo o sistema, sujeitando-se a perder todo o tempo e custo envolvidos caso o desempenho desejado não seja atingido. A segunda opção é o emprego de protótipos. A construção de protótipos apresenta um custo menor de desenvolvimento em relação à construção do sistema, adiantando algumas informações sobre o comportamento do futuro sistema [ST87].

Outra aplicação dos protótipos é na verificação de alternativas para o sistema [ST87]. Quando o objetivo for analisar o comportamento de um sistema mediante algumas alterações, é impraticável que essas alterações sejam feitas diretamente sobre o sistema.

Apesar das vantagens de menor custo e de maior facilidade de alteração em relação à construção e alteração do sistema real, o protótipo também se constitui como

uma ferramenta cara (quanto maior o custo de desenvolvimento do protótipo maior o custo das alterações [ST87]), que consome uma quantidade considerável de tempo [NU78, SA90,SA90a]. Outra dificuldade é encontrada na abstração das características essenciais do sistema, que serão refletidas no protótipo. Muitas vezes o sistema real encontra-se em um estágio de desenvolvimento em que ainda não estão bem definidas diversas características sobre sua configuração e operação, inviabilizando a construção de um bom protótipo [HA88].

Recomenda-se o desenvolvimento de protótipos como ferramenta para verificação do projeto final ou então para verificação de um sistema existente, onde poucas modificações são necessárias [NU78, SA90, SA90a]. O desenvolvimento de protótipos não é garantia de sucesso, porém essa estratégia possibilita detectar erros conceituais em um menor tempo e custo possíveis [ST87].

O desenvolvimento de um protótipo pode ser dividido em cinco etapas [CA87, HA88]: (figura 4.2)

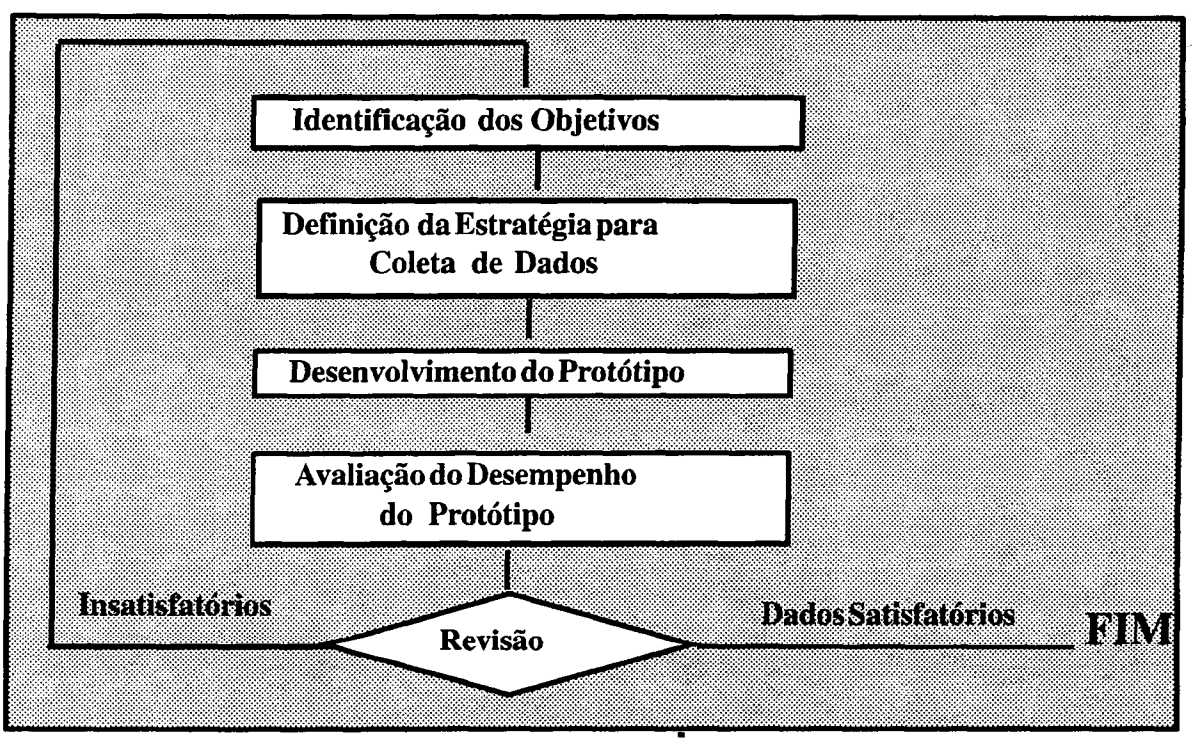

Figura 4.2 Etapas de Desenvolvimento de um Protótipo

A primeira etapa corresponde à identificação dos objetivos que determinaram a construção do protótipo e também à definição das características essenciais do sistema em questão, com ênfase para as medidas de desempenho que deverão ser analisadas. Durante essa etapa deve-se, também, estabelecer uma estimativa de tempo e de custos para que se tenha o protótipo em operação.

A definição da estratégia para coleta de dados que alimentarão o protótipo, assim como os passos a serem seguidos para o desenvolvimento do protótipo correspondem à *segunda etapa.* Essa etapa é muito importante pois informações infundadas podem determinar conclusões erradas sobre o desempenho do sistema, implicando na construção de um sistema que em operação não corresponde à análise ou então implicando no abandono de um projeto que poderia corresponder às expectativas.

A terceira etapa consiste no desenvolvimento do protótipo, considerando-se os requisitos definidos durante a primeira etapa.

Durante *a quarta etapa* o sistema é avaliado, obtendo-se as medidas de desempenho e analisando-as. A avaliação deve ser minuciosa ponderando-se a limitação do protótipo e estendendo-se os resultados para o sistema real.

A quinta etapa corresponde à revisão do protótipo. Independentemente do resultado da análise a revisão deve ser efetuada de forma a verificar se todos os objetivos foram atingidos e se as características essenciais foram reproduzidas no protótipo. Se não houver confiança nos resultados obtidos pela análise deve-se reiniciar todo o processo, tendo como ponto de partida a primeira etapa. Caso contrário o desenvolvimento ou não do novo sistema é decidido baseado nas informações obtidas pela análise através do protótipo.

4.3 - Benchmarks

Um benchmark é um programa escrito em linguagem de alto nível, sendo considerado como representativo para um dada classe de programas de aplicação. Programas de ordenação e programas de atualização de arquivos são exemplos típicos dessa ferramenta [TA92]. Os resultados são altamente dependentes da natureza dos benchamarks, por exemplo: o resultado de um benchmark que envolve um grande número de cálculos da U.C.P. pode ser diferente do resultado de um benchmark que consiste em percorrer um único arquivo [TA92].

Um programa benchmark é executado no sistema proposto, a comparação do tempo total de execução fornece uma boa indicação do desempenho do sistema. A qualidade dos compiladores também pode ser obtida, medindo-se tanto o tempo de compilação como o tempo de execução de cada benchmark [TA92]. O mesmo programa pode ser codificado e executado em linguagem de máquina e em linguagem de alto nível para avaliar a velocidade do código de saída do compilador [LU71].

Os benchmarks são indicados para a comparação de diferentes sistemas antes da aquisição de um novo. Podem também auxiliar no projeto de um novo sistema, codificando os módulos e executando-os sobre outras máquinas, mas essa experiência fica limitada quando devem ser previstas quais rotinas serão representativas do comportamento do novo sistema [LU71].

Para a comparação entre diferentes máquinas utilizava-se a unidade MIPS ("Millions of Instructions Per Second"). Atualmente essa unidade perdeu seu sentido [WE90]. Esse fato tornou-se mais evidente com o surgimento das máquinas RISC ("Reduced Instruction-Set Computer") , pois uma operação realizada por uma máquina CISC ("Complex Instruction-Set Computer") geralmente requer várias instruções RISC. Devido a esse problema a unidade MIPS tem sido constantemente redefinida como por exemplo "VAX MIPS". Nesse caso MIPS é somente um fator de desempenho, dado por uma máquina, relativo ao desempenho do VAX 11/780. Se uma máquina executa algum programa ou conjunto de programas x vezes mais rápido que um VAX 11/780 é chamada máquina X-MIPS. Recomenda-se utilizar a velocidade desses programas benchmark como fator comparativo, ao invés dos MIPS [WE90].

Diz-se que o melhor benchmark é a própria aplicação do usuário. Isso nem sempre é real pois não é sempre viável executar a aplicação em todas as máquinas em questão.

O melhor benchmark, no entanto, é definido como aquele que satisfaz às seguintes condições [WE90]:

(1) - O programa é escrito em linguagem de alto nível, sendo possível executa-lo em qualquer máquina.

(2) - O programa deve ser representativo de algum estilo de programação (numérico, comercial).

(3) - O programa pode ser avaliado facilmente.

(4) - O programa possui larga distribuição.

Notam-se que os requisitos acima são contraditórios. Quanto mais representativo for o programa benchmark mais complicado o será. Dessa forma aumenta o grau de dificuldade para a avaliação e os resultados podem ficar disponíveis somente para algumas máquinas. Isso justifica a popularidade de alguns benchmarks que não são programas de aplicação completos mas são representativos para uma certa área de aplicação [WE90].

A seguir serão discutidos alguns benchmarks: Whetstone, Dhrystone e Linpack, cujos resultados são largamente referenciados nas publicações dos fabricantes. As tabelas (A-D) fornecem informações sobre as características desses benchamarks em linguagens de alto nível [WE90].

Sentença	Dhrystone	Whetstone	Linpack
Sentença FOR (avaliação)	6.8	17.3	49.3
Sentença Goto	-	0.5	-
Sentença While (avaliação)	4.9	-	-
Sentença Switch	1.0	-	-
Sentença Break	1.0	-	-
Sentença Return (com expressão)	4.9	-	-
Sentença Call (procedimento do usuário)	9.7	11.9	-
Sentença Call (Função do usuário)	4.9	-	-
Sentença Call (Procedimento do sistema)	1.0	-	-
Sentença Call (função do sistema)	1.0	4.7	-
Sentenças IF	13.6	8.6	2.2
Associações	51.6	56.7	48.5
Total	100.0	100.0	100.0

Tabela A - Distribuição em Porcentagem das Sentenças

Operador	Dhrystone	Whetstone	Linpack
+ (int/char)	21.0	11.0	14.1
- (int)	5.0	6.0	-
* (int)	2.5	6.0	-
/ (int)	0.8	-	-
+(float/double)	-	14.9	14.1
- (float/double)	-	2.1	-
* (float/double)	-	9.3	14.1
/ (float/double)	-	4.6	-
<, ? <= ??	10.1	10.7	14.5
>, >=	11.7	2.8	0.6
Operadores Lógicos	3.3	-	0.2
Indexação (unidimensional)	5.9	24.5	-
Indexação (bidimensional)	3.4	-	0.2
Seleção de Registro	7.6	-	42.3
Operadores Específicos do C	13.4	7.2	-
Total	100.0	100.0	100.0

Tabela B - Distribuição em Porcentagem do Operador

Tipo de Operando	Dhrystone	Whetstone	Linpack
Inteiro	57.0	55.7	67.2
Caracter	19.6	-	-
Float / Double	-	44.3	32.8
Enumerado	10.9	-	-
Lógico	4.2	-	-
Vetor	0.8	-	-
Cadeia de Caracteres	2.3	-	-
Ponteiro	5.3	-	-
Total	100.0	100.0	100.0

Tabela C - Distribuição em Porcentagem dos Tipos de Operandos

Tipo de Operando	Dhrystone	Whetstone	Linpack
Local	48.7	0.4	49.5
Global	8.3	56.3	-
Parâmetro (Valor)	10.6	18.6	17.0
Parâmetro (Referência)	6.8	1.9	24.6
Resultado de Função	2.3	1.3	-
Constantes	23.4	21.6	8.8
Total	100.0	100.0	100.0

Tabela D - Distribuição em Porcentagem da Localidade dos Operandos

4.3.1 Principais Benchmarks

WHETSTONE

Foi o primeiro programa na literatura projetado especificamente para ser um benchmark. Publicado em 1976, sendo Algol 60 a linguagem original de publicação, atualmente é usado quase que exclusivamente na versão em Fortran [WE90].

O programa consiste em diversos módulos, cada um contendo sentenças de algum tipo particular (aritmética inteira, aritmética em ponto flutuante, sentenças se, chamadas a procedimentos e outras), terminando com sentenças para impressão dos resultados [WE90]. Pesos são atribuídos aos diferentes módulos tal que a distribuição das "instruções Whetstone" para o programa benchmark corresponda à distribuição observada no programa amostral. Esses pesos são escolhidos de tal forma que o

programa execute um múltiplo de um milhão dessas instruções Whetstone; assim os resultados são fornecidos como KWIPS ("kilo Whetstone Instructions per Second"). Dessa forma o termo "instruções por segundo" é mantido, mas impõe um sentido independente de máquina.

O programa possui alta porcentagem de dados e operações em ponto flutuante. Isso ocorre porque o objetivo do benchmark é justamente representar programas numéricos.

Uma alta porcentagem do tempo de execução é gasto com funções de bibliotecas matemáticas. Essa propriedade é derivada dos dados estatísticos que constituem a base do Whetstone.

Como pode ser observado na tabela D o Whetstone usa poucas variáveis locais e muitas variáveis globais. Um compilador no qual as variáveis globais mais usadas são alocadas em registradores, estimulará o desempenho do whetstone [WE90].

LINPACK

O programa Linpack foi publicado em 1976 [WE90]. Correspondia inicialmente somente a uma coleção de subrotinas de álgebra linear freqüentemente usadas em programas Fortran. O autor considerou o que era parte da "vida real" de um programa e o transformou em benchmark.

O programa opera sobre uma grande matriz, onde as subrotinas internas manipulam-na como um vetor unidimensional (uma otimização comum na programação de Fortran) [WE90]. Os resultados são fornecidos em milhões de operações em ponto flutuante por segundo (MFLOPS); o número de operações em ponto flutuante que o programa executa pode ser derivado a partir do comprimento do vetor.

Essa terminologia implica que as operações inteiras são negligenciadas, ou ainda, que seu tempo de execução está incluído nas operações em ponto flutuante [WE90].

Para o programa Linpack é necessário conhecer os seguintes pares de atributos [WE90]:

- **simples x duplo**. Se a precisão é simples ou dupla para dados em ponto flutuante.

- **"Rolled" x "Unrolled"**. Na versão "unrolled" os laços ("loops") são otimizados; o índice do laço é incrementado de 4 e o corpo do laço contém quatro grupos de sentenças para os índices i, i+1, i+2, i+3. Essa técnica economiza tempo de execução para a maioria das máquinas e compiladores. As máquinas vetoriais mais sofisticadas, todavia, onde o laços "unrolling" é feito na geração de códigos para hardware vetorizado, usualmente executa a

versão padrão ("rolled") mais rápido.

- **SALB codificado x SALB Fortran**. O programa Linpack depende P altamente do pacote de Subrotinas de Álgebra Linear Básicas (SALB). O SALB codificado significa que essas subrotinas foram rescritas em linguagem de máquina. Computando-se a razão do tempo de execução entre o SALB codificado e as versões Fortran para a mesma máquina tem-se uma noção sobre a qualidade de otimização de código de compilador Fortran.

O programa Linpack possui alta porcentagem de operações em ponto flutuante.

Outra característica importante do Linpack é a utilização de variáveis locais opostamente ao Whetstone (Tabela D).

DHRYSTONE

O Dhrystone foi publicado em 1984 sendo ADA sua linguagem de publicação [WE90]. Atualmente é usado principalmente na linguagem C.

A base para o Dhrystone é a distribuição das características da linguagem fonte na programação não numérica - programação tipo-sistema - como por exemplo sistema operacional, compiladores, editores, etc. Além da diferença nos tipos de dados, os programas numéricos e os programas tipo-sistema têm duas diferenças: os programas tipo-sistemas contêm laços menores, sentenças simplificadas de cálculos e mais sentenças "SE" e chamadas a procedimentos [WE90].

O Dhrystone consiste em doze procedimentos incluídos em um laço de medidas com 94 sentenças. Um laço corresponde a um Dhrystone, os resultados são usualmente dados em Dhrystone por segundo.

Entre as principais características podem ser citadas: o Dhrystone não contém operações em ponto flutuante em seu laço de medida; uma porcentagem considerável do tempo de execução é gasto em funções de cadeias de caracteres. Somente uma pequena quantidade de dados globais é manipulada [WE90].

4.3.2 Outros Benchmarks

Benchamarks EDN.

A coleção de programas conhecida como *EDN* foi publicada em 1981. Originalmente os programas foram escritos em linguagem de máquina (8086, 68000 e Z8000); com o objetivo de avaliar a velocidade dos microprocessadores, sem avaliar a

qualidade do compilador [WE90].

Um subconjunto dos benchmarks originais é atualmente usado na versão C:

Benchmark E: busca em cadeia de caracteres.

Benchmark F: bit teste/set/reset.

Benchmark H: Inserção em lista encadeada.

Benchmark I: Quicksort.

Benchmark K: Transformação de matriz.

Esse subconjunto dos benchmarks EDN tem sido usado para comparação dos processadores RISC e CISC [WE90].

Benchmark SPEC

O SPEC - Cooperativa para Avaliação da Performance dos sistemas ("the systems performance evaluation cooperative effort") - surgiu do princípio de que os benchmarks existentes (consistindo de programas pequenos) eram inadequados [WE90]. Quando um programa pequeno torna-se popular como um benchmark, os escritores de compiladores ficam inclinados a direcionar seus compiladores para otimizações que beneficiam particularmente esse benchmark - por exemplo otimizações em cadeias de caracteres para o Dhrystone.

A meta do SPEC é coletar, padronizar e distribuir programas de aplicações que podem ser usados como benchmark.

4.3.3. Comentários Finais Sobre Benchmarks

Na avaliação de um benchmark deve-se estimar o efeito da programação no desempenho do benchmark - tarefa difícil - além de outros fatores como: a influência do compilador, o sistema de biblioteca em tempo de execução e o tamanho da memória cache.

Apesar da existência de compiladores sofisticados e otimizados os pequenos benchmarks continuarão sendo utilizados por algumas razões, entre as quais destacam-se:

a. Os fabricantes, algumas vezes, baseiam sua taxa de MIPS naquelas fornecidas por esses benchmarks. Um exemplo é a decisão da IBM de basear-se no número de MIPS da versão 1.1 do Dhrystone. Subseqüentemente, DEC e Motorola também

alteraram seus valores de MIPS para os valores da versão 1.1 do Dhrystone, para ficarem competitivos com a IBM [WE90].

b. Os benchmarks de "vida real" não estão completamente livres de distorções a partir de efeitos imprevistos ou efeitos colaterais. As experiências adquiridas com os pequenos benchmarks podem auxiliar na identificação desses efeitos [WE90].

O maior problema dos benchmarks é que um dado benchmark pode não ser representativo do programa do usuário. É difícil concluir quantas séries de testes devem ser necessárias para garantir uma proporção adequada de compilação versus execução, tipos ou classes de programas, requisitos de equipamentos e considerações sobre prioridades [LU71]. Outra limitação é que os benchmarks não constituem representações significativas para sistemas interativos, a menos que existam facilidades adicionais que simulem os terminais do sistema [TA92].

4.4 - Coleta De Dados

Coleta de dados é a ferramenta, entre todas as técnicas de desempenho abordadas, que fornece o valor mais preciso para as medidas de desempenho [KO78, SA90, SA90a, FE92]. A aplicação dessa técnica, assim como as demais técnicas de aferição, só é viável em sistemas que estão em operação.

Dessa forma, quando o objetivo for unicamente a análise de desempenho de um sistema computacional, a ferramenta indicada para essa análise é a coleta de dados diretamente do sistema. Essa coleta não deve interferir no comportamento do sistema, o que pode trazer consequências desastrosas à análise, pois o tempo gasto para coletar os dados poderiam ser incluídos nos cálculos das medidas de desempenho, gerando interpretações enganosas sobre o desempenho do sistema.

Outra motivação para a coleta de dados é estimar os parâmetros de entrada requisitados por modelos analíticos e/ou de simulação [KO78]. Essa opção é adequada quando o objetivo da análise for a avaliação do sistema e também quando o objetivo for a verificação de algumas alternativas para o sistema [KO78]. Isso ocorre porque mesmo depois do sistema já em operação a adoção de modelos prognósticos constitui-se como uma importante peça na melhoria do desempenho e no planejamento de sua capacidade. Considerando-se, por exemplo, que um gerente de um centro computacional deseja melhorar a capacidade de serviço do centro e dispõe para isso das opções: trocar a U.C.P. com uma de maior potência, expandir a capacidade da memória principal, aumentar o número de caminhos ("paths") para dispositivos secundários de armazenamento ou ainda trocar alguns algoritmos de alocação de recurso. Reconfigurar o sistema para examinar cada uma dessas opções é impraticável. Assim, o sistema pode ser modelado e resolvido por simulação ou métodos analíticos e as entradas para esse modelo podem ser abstraídas dos dados

coletados no sistema em operação.

Uma tarefa, freqüentemente negligenciada, em que a coleta de dados também pode ser empregada, é a validação do modelo de simulação ou do modelo analítico [KO78]. No caso do estudo de alternativas para um sistema, pode ser elaborado inicialmente um modelo correspondente ao sistema em questão. O objetivo de desenvolver esse modelo é criar uma base comparativa e confiável para a inserção das alternativas desejadas. Assim, após o modelo ser resolvido, os dados obtidos podem ser comparados com os dados coletados diretamente do sistema e se uma dada precisão for atingida pode-se concluir que o modelo é uma representação válida do sistema real. As alterações podem então ser refletidas no modelo e assim que for solucionado, seus resultados podem ser comparados com os dados abstraídos do sistema real, fornecendo parâmetros confiáveis para a análise.

4.4.1. Técnicas Para Coleta de Dados

As técnicas para coleta de dados podem ser agrupadas em duas classes [KO78]:

- Técnicas de Instrumentação de Hardware (usando monitores de hardware).
- Técnicas Programadas de Coleta de Dados (utilizando monitores de software).

Os monitores de hardware são compostos, tipicamente, dos seguintes itens [KO78]:

1 - Conjunto de sensores eletrônicos ou "probes".

2 - "Plugboards" lógicos (pontas lógica para monitorar os sinais).

3 - Um conjunto de contadores.

4 - Um "display" (mostrador).

5 - Uma unidade de armazenamento.

O conjunto de sensores eletrônicos monitora o estado dos pontos terminais escolhidos. Esses pontos usualmente ficam em recursos como a U.C.P., canais de transmissão e dispositivos periféricos. Um sinal detectado pelo sensor eletrônico da U.C.P. pode indicar, por exemplo, se está ocupada ou ociosa em um dado instante. Os sinais monitorados, que são tipicamente sinais binários, são processados em um circuito lógico para transformar ou selecionar um fluxo de sinais em um conjunto de sinais interpretáveis. Os contadores são usados para cálculos estatísticos: para contar

um número de ocorrências de eventos específicos ou o tempo de duração de uma atividade do sistema. A saída dos dados pode ser feita da seguintes formas: o conteúdo do contador pode ser periodicamente mostrado em tempo real em um terminal gráfico ou podem ser escritos em fitas magnéticas para processamento posterior.

Em alguns monitores é incluído um minicomputador que atua como um controlador e processador do monitor de hardware [K078]. Isso permite que o monitor de hardware tenha capacidade adicionais, tais como geradores de histogramas. A inclusão de um microcomputador possibilita, ainda, que a saída do monitor seja conectado a periféricos, incluindo discos e "displays". O microcomputador pode também facilitar modificações "on-line" de medidas e experimentos, tal que o analista possa conduzir a coleta de dados efetivamente.

Monitores de Software

Como discutido acima, o monitor de hardware pode coletar informações sobre o uso de recursos e também coletar informações sobre várias atividades sem perturbar o sistema. A limitação dessa técnica está associada à dificuldade de captar os eventos relacionados com o software, como por exemplo verificar a existência ou não de uma fila de espera associada a um recurso do sistema. Na análise de desempenho é importante observar as sequências de mudanças de estado que ocorrem no sistema [KO78]. Essas informações são especialmente úteis para o estudo de como os vários componentes do sistema operacional (por exemplo: a política de escalonamento da U.C.P., a política de alocação de memória) afetam o desempenho do sistema.

Esse tipo de informação pode ser obtido através de monitores de software. Esses monitores podem ser divididos em dois grupos: monitores orientados a eventos e monitores de amostragem [KO78].

Um monitor orientado a evento corresponde a uma coleção de rotinas do sistema que são executadas após a ocorrência de interrupções da U.C.P. ou após a ocorrência de alguns eventos. O monitor de software pode também coletar informações sobre o status do sistema percorrendo as várias tabelas e filas dos vários eventos do sistema. Assim, um monitor de software pode medir o comprimento de fila de um dispositivo, a freqüência de acesso a esses dispositivos entre outras informações. Informações sobre a utilização de memória são obtidas mais facilmente usando técnicas de software do que de hardware.

A versatilidade e diversidade das informações obtidas por monitores de software acrescentam uma sobrecarga ao sistema: a cada interrupção ou ocorrência de um evento a rotina de coleta de dados é executada, o que certamente consome tempo de U.C.P. , ocupa espaço na memória principal e utiliza canal para transferência dos dados [KO78]. Dessa forma a adoção de um monitor de software orientado a eventos pode acarretar a degradação do desempenho do sistema durante o processo de monitoramento.

Esse problema pode ser amenizado, em alguns casos, com o uso de técnicas de amostragem [KO78]. O monitor de amostragem chama a rotina de monitoramento em intervalos regulares para permitir que o processador seja interrompido por um temporizador interno. Nesses instantes o programa monitor inspeciona o conteúdo das tabelas (residentes na memória), que contém informações sobre o uso dos recursos e atividades do sistema. Ou seja, o monitor de amostragem utiliza fotografias ("snapshots") do estado do sistema em intervalos periódicos. A razão de amostragem, em geral, pode ser baseada na razão de mudanças de estado; na precisão desejada na estimativa dos parâmetros das distribuições e no custo envolvido para obtenção das medidas.

A desvantagem dessa técnica está no fato de que alguns eventos não são capturados.

4.5 - Comentários Finais

Cada uma das técnicas discutidas neste capítulo é indicada para uma proposta de avaliação. Para se optar entre uma delas deve-se considerar o objetivo da análise, custo e tempo envolvidos na avaliação de desempenho.

Bibliografia

[CA87] - Castro, J.M., 'Prototipação! Uma Proposta de Utilização", XX Congresso Nacional de Informática, 1112-1118, São Paulo, 1987.

[DA89] - Davies, Ruth and O'Keef, Robert, "Simulation Modeling With-Pascal, Prentice Hall International Ltda, London, 1989.

[ED92] - Edwards G., Sankar R., "Modeling and Simulation of Networks using CSIM - Simulation", vol 58(2), February 1992.

[FE92] - Fernandes, M. M., "Modelagem Analítica de Desempenho de Sistemas Multiprocessadores: Aplicação ao Multiprocessador CPER, Dissertação de mestrado, Universidade Federal de São Carlos, Dezembro de 1992.

[HA88] - Harker, S., " The Use of Prototyping and Simulation in the Development of Large-Scale Applications", The Computer Journal, Vol. 31(5), 1988.

[KO78] - Kobayashi, H., "Modeling and Analysis - An Introduction to System Performance Evaluation Methodology", Addison-Wesley Publishing Company, 1978.

[LU71] - Lucas, Jr. H. C., "Performance Evaluation and Monitoring", Computing Surveys, Vol. 3(3), September 1971.

[MA75] - MacDougall, M. H., "System Level Simulation", em Digital System Design Automation: Languages, Simulation & Database, Cap 1, Computer Science Press Inc., 1-115,1975.

[MA80] - Maryansky, Fred J., "Digital Computer Simulation", Haydem Book Company Inc. 1980.

[MA87] - MacDougall, M. H., "Simulation Computer Systems - Techniques and Tools", The MIT Press, 1987.

[MO86] - Moura, J. A. B. & Sauve, J. P. & Giozza & Araújo, J. F. M., "Redes Locais de Computadores - Protocolos de Alto Nível e Avaliação de Desempenho", McGraw-Hill, 1986.

[NU78] - Nutt, G. J., "A Case Study of Simulation as a Computer Design Tool", Computer, 31-36, October 1978.

[PE91] - Pegden, C.D. et al, Introduction to Simulation Using SIMAN", MacGraw-Hill International Editions, 1991.

[SA84] - Sargent, Robert G. - " A Tutorial on Verification and Validation of Simulation Models", Proceedings of the 1984 Winter Simulation Conference, pag 115-121.

[SA85] - Saydam, T. - "Process Oriented Languages", Simuletter, vol 16(2), April 1985.

[SA90] - Santana, M. J., "An Advanced Filestore Architecture for a Multiple - Lan Distributed Computing System", PHD Thesis, University of Southamptom, January 1990.

[SA90a] - Santana, R. H. C., "Performance Evaluation of Lan-Based File Servers", PHD Thesis, University of Southamptom, January 1990.

[SH88] - Sharma, R. and Rose, L. I., "Modular Design for Simulation", Software, Practice and Experience, Vol. 18(10), 1988.

[SO92] - Soares, L. F. G., "Modelagem e Simulação Discreta de Sistemas", Editora Campus Ltda, 1992.

[SO92a] - Souza, R. C. G., "Desenvolvimento de uma Extensão Funcional em C para a Construção de Um Ambiente de Simulação Orientado a Processos", Relatório de Iniciação Científica, ICMSC-USP, 1992.

[SP92] - Spolon, R., "Extensão Funcional de Módula-2 para Simulação de Sistemas Discretos", Relatório de Iniciação Científica, ICMSC-USP, 1992.

[ST87] - Stus, J. M., "Considerações Sobre o Desenvolvimento Por Protótipos", XX Congresso Nacional de Informática, 1044-1047, São Paulo, 1987.

[TA92] - Tanenbaum, Andrew S., "Modern Operating Systems", Prentice-Hall International Editions, 1992.

[WE87] - Westphall, C. B. e Tarouco, L. M. R. e Wagner, J. B., "Medidas Para Avaliar Desempenho Em Redes", XX Congresso Nacional de Informática, Vol. 2, 722-726, São Paulo, 1987.

[WE90] - Weicker, R. P., "An Overview of Common Benchmarks", IEEE Computer, 65-75, December 1990.