CAPÍTULO 2

MODELAGEM PARA AVALIAÇÃO DE DESEMPENHO E GERAÇÃO DE SEQÜÊNCIAS DE TESTES

Nandamudi L. Vijaykumar

RESUMO

Validação e Verificação de Sistemas tem se tornado um tópico muito importante nos últimos anos. Este tópico engloba, entre outras, áreas como Avaliação de Desempenho e Testes. A intenção do curso é dar uma breve introdução a estas áreas, descrevendo alguns conceitos básicos. O objetivo do curso é também mostrar como se obtêm informações (ou medidas) sobre desempenho de um dado sistema, além de comentar como se geram seqüências de testes para poder fornecer um veredicto de um determinado software. No entanto, o foco principal do curso é tratar o tópico da modelagem de sistemas, a partir da qual se podem determinar medidas de desempenho e obter casos ou seqüências de testes. Em particular, a modelagem descrita com maiores detalhes é a técnica de especificação Statecharts.

Parte A: Modelagem de Sistemas Reativos para Avaliação de Desempenho (em colaboração com Carlos Renato Lisboa Francês, UFPA, Belém, PA)

2.1 Introdução

No mundo moderno, a dependência da tecnologia tem sido uma realidade, e este fator tem crescido ainda mais com a redução de custo na manufatura de sistemas computacionais. Esta tecnologia está presente em várias áreas, como aparelhos domésticos, sistemas bancários, automóveis, além daquelas consideradas áreas críticas, como controle de tráfego aéreo, sistemas médicos, sistemas de comunicação, aviação, aplicações espaciais, entre outras. Naturalmente, esta vantagem em não só popularizar a tecnologia, mas também utilizá-la de forma efetiva, leva a uma maior complexidade nos hardwares e softwares desenvolvidos. Esta complexidade embutida, em particular em sistemas críticos, infelizmente não está livre de erros. Erros em sistemas críticos têm conseqüências graves, podendo causar prejuízos financeiros, além de colocar em perigo vidas humanas. Isto implica na obrigação de construir tais sistemas críticos com altos índices de qualidade e confiabilidade. Para atingir este objetivo, é necessário investigar técnicas e metodologias para melhorar o processo de desenvolvimento e construção de sistemas, mesmo aqueles não críticos. Com o uso de computadores cada vez mais freqüente, é mais que natural que o processo deva ser automatizado. Neste sentido, as técnicas e metodologias para melhorar o processo de desenvolvimento devem ser formais, para que se possa aplicar um tratamento computacional.

A pergunta é como obter altos índices de qualidade e confiabilidade nos sistemas a serem desenvolvidos, ou seja, como se garante que, antes mesmo da sua implementação ou construção, um sistema poderá ser avaliado e validado quanto ao seu comportamento e funcionamento. Para isto, existe a área, hoje considerada um

importante tópico de pesquisa, conhecida como *avaliação de desempenho*. Quais, então, seriam as técnicas ideais a serem utilizadas para avaliar o desempenho?

Modelos de desempenho referem-se aos sistemas cuja avaliação é estudada e analisada. Estes modelos são, em geral, estocásticos. A palavra *estocástico* vem do termo grego *stochos* (στοχαστικοξ), que significa *adivinhar*; ou seja, a característica desses modelos é a aleatoriedade. Pode-se dizer que o comportamento de um modelo estocástico não é determinístico.

Nestes modelos, o comportamento do sistema é geralmente representado como um sistema orientado a eventos, conhecidos como sistema reativo. Os sistemas reativos são aqueles que respondem a estímulos internos e externos, ou seja, são organizados em estados que, quando há alguma perturbação (através de um estímulo) são modificados. Um exemplo simples de tais sistemas seria o processador de um computador que está em estado *Ocioso* e, quando chega uma tarefa a ser processada, tem seu estado modificado para *Processando*. Nesse caso, a chegada de uma tarefa é considerada um estímulo que perturbou o sistema modificando o seu estado *Ocioso* para *Processando*.

O documento descrito aqui na Parte A aborda o tópico de avaliação de desempenho, o que inclui as soluções existentes para a sua quantificação. Discute também a modelagem de sistemas em geral, um importante tópico associado a esta técnica, já que a idéia é obter os parâmetros de desempenho bem antes de o sistema estar pronto e lançado no mercado. O conceito de sistemas tratados no contexto deste documento é relativo a sistemas reativos considerados sistemas complexos. O documento está organizado da seguinte maneira: a Seção 2 descreve a avaliação de desempenho e as soluções que podem ser utilizadas para gerar esta medida, além de discutir como a modelagem pode ser associada a estas soluções. Algumas técnicas de modelagem como Redes de Filas, Redes de Petri e Statecharts também serão brevemente comentadas neste capítulo. A Seção 3 descreve a ferramenta PerformCharts, baseada na técnica de modelagem Statecharts [Harel, 1987] e utilizada na avaliação de desempenho. Finalmente, algumas considerações e comentários serão descritos na Seção 4.

2.2 Avaliação de Desempenho

Dentro do ciclo de desenvolvimento de um produto, especialmente na fase de projeto, ter em mãos a quantificação de parâmetros como desempenho e confiabilidade é uma grande vantagem, que, de certa forma, possibilita desenvolver sistemas como de computação, comunicação, manufatura e outros com um grau significativo de qualidade. A quantificação de tais parâmetros permite avaliar o desempenho desses sistemas, através da aplicação de métodos probabilísticos [Bolch et al., 1998]. A avaliação de desempenho tem o objetivo de prever quantitativamente o comportamento de um sistema, e pode ser empregada sempre que houver a necessidade de calibrar ou adaptar sistemas já existentes. A previsão do comportamento pode calcular os impactos das adaptações ou mudanças sobre o desempenho do sistema [Haverkort, 1998].

A avaliação de desempenho praticamente consiste em obter medidas de desempenho através do monitoramento do sistema. Através dela, é possível detectar quais operações dentro do sistema demandam mais tempo que outras, além de ter uma visão sobre quais componentes estão sobrecarregados. Essas medidas dependem da disponibilidade do sistema. Nesses casos, aplicam-se técnicas que lidam com experimentação, que podem ser consideradas técnicas diretas, pois a avaliação de desempenho é realizada num sistema real. Ou seja, são realizados experimentos em sistemas já existentes e, com base nos resultados obtidos, mudanças são implementadas nestes sistemas, melhorando assim a sua qualidade e aumentando seu grau de confiabilidade. Por exemplo, para se ter uma idéia do comportamento de um hardware, é necessário que ele esteja disponível. O grande problema é que nem sempre isso é possível, pois deve ser considerado o fator custo envolvido na

aquisição do equipamento. Portanto, devido a fatores como custo e disponibilidade do sistema, nem sempre a avaliação de desempenho pode ser aplicada.

Mas, mesmo quando isso não é possível, deve haver uma maneira de avaliar o desempenho de um sistema ainda na fase de projeto, para que se possa predizer se o projeto satisfaz os requisitos especificados e, assim, possibilitar realização de mudanças, levando em consideração o custo-benefício associado a elas. Nestas situações, entram em cena as técnicas que lidam com modelagem, também conhecidas como avaliação de desempenho baseada em modelos (*model-based performance evaluation*). O processo, neste caso, deve obedecer ao seguinte roteiro: se o sistema não estiver disponível, deve-se ter uma descrição não-ambígua do mesmo e, a partir dela, construir o modelo abstrato, que deve se basear em conceitos matematicamente bem definidos em termos de seus componentes e suas interações. Deve também definir claramente a interação com o ambiente externo. Infelizmente, não há uma *receita* para se construir um modelo abstrato e, em especial, o modelo para avaliar o desempenho.

São criadas abstrações de sistemas a serem avaliados e aplicadas a elas algumas soluções a partir das quais é possível predizer o desempenho do projeto. As soluções podem ser por meio de simulação ou abordagem analítica.

Mas, antes de apresentar como são utilizadas estas soluções, vale um comentário sobre as métricas de desempenho que interessam aos usuários ou projetistas. É importante salientar que estas métricas devem ser capazes de fornecer respostas às perguntas que existem sobre o sistema em análise. Alguns autores, como [Haverkort, 1998], classificam estas métricas nas seguintes categorias:

- Métricas dos Clientes do Sistema: visão do usuário
- Tempo de Resposta (*Job Response Time*)
- Vazão de Clientes (*jobs*)
- Tempo de Espera de Cliente
- Tempo de Atendimento do Cliente (*Job Service Time*)
- Métricas do Sistema propriamente dito: organização interna do sistema a ser avaliado
- Número de Clientes no Sistema
- Utilização dos Recursos do Sistema

Construído o modelo, este deve ser analisado para poder obter as métricas. A condução de análise de desempenho deveria, sempre que possível, contar com alguma ferramenta, tanto na construção do modelo quanto na aplicação de alguma técnica de solução na modelagem. Os sistemas modernos são complexos e exigem que as técnicas de modelagem possuam elementos capazes de descrever encapsulamento, atividades paralelas e sincronização e comunicação entre os componentes paralelos. Exemplos já mencionados destes sistemas modernos são sistemas de computação, comunicação, manufatura, aplicações espaciais, entre outros. Em outras palavras, pode-se dizer que não é uma tarefa fácil a modelagem do sistema complexo a ser avaliado. Outro fator que justifica a necessidade de contar com alguma ferramenta computacional é a dificuldade de analisar o modelo simplesmente por meio de uma calculadora ou uma planilha. Em relação às técnicas de solução, pode-se classificá-las em duas categorias: técnicas de simulação e técnicas analíticas.

As técnicas de simulação abrangem as classes mais gerais possíveis de modelos. Nesse tipo de solução, o comportamento do sistema (em avaliação) é *imitado* executando-se um programa de simulação. A vantagem de se utilizar simulação está na sua aplicação a qualquer problema, e no fato da representação poder chegar aos detalhes que se deseja capturar do sistema em questão. Outra vantagem é a existência de vários pacotes de

software e linguagens de programação que permitem a construção e execução destes modelos de simulação. No entanto, o tempo para executar tais modelos é grande, em especial quando se trata de sistemas de grande porte, e ainda maior quando é necessária uma precisão maior dos resultados.

A outra alternativa de solução é por meio de abordagem analítica. Esta abordagem pode ser viabilizada com base no fator custo-benefício do seu uso. Estes modelos analíticos (ou estocásticos) são considerados modelos que lidam com um espaço de estados. Os modelos mais populares e bastante utilizados são as Cadeias de Markov. Nestes modelos de espaço de estados, a desvantagem está na necessidade de assumir fatores que nem sempre refletem a realidade, para que possam ser tratáveis.

As próximas seções apresentam uma breve introdução às técnicas de simulação e às soluções por abordagem analítica através de Cadeias de Markov.

2.2.1 Simulação

A vantagem dos modelos é possibilitar a previsão de como se comporta um sistema bem antes dele ser construído. Não há uma solução *ideal* para resolver esta previsão. Depende muito das características do sistema a ser avaliado, como por exemplo, número de informações que este sistema deve manipular [Francês, 1998]. A vantagem de se utilizar a simulação está no fato de não haver restrições sobre os tipos de modelos criados cujo desempenho deve ser avaliado. A abordagem por simulação para resolver o modelo parece mais adequada para praticamente todos os sistemas, em especial quando se compara aos métodos analíticos, que assumem algumas características do sistema mesmo que elas não representem a realidade. No entanto, é preciso lembrar que, ao utilizar simulação, poderão existir restrições em relação ao tempo de processamento do computador ou à quantidade de memória.

Um dado sistema pode ser avaliado de acordo com a natureza das mudanças de estado deste sistema. A mudança de estado, em geral, ocorre em função do tempo, que é a variável independente, e poderá ser discreta ou contínua [Soares, 1992]. Então, a simulação pode ser classificada como segue [Haverkort, 1998].

- Simulação de eventos contínuos: é aplicada em casos onde os estados mudam continuamente com o tempo. As variáveis dependentes variam continuamente ao longo do tempo simulado, como mostra a Figura 2.1a. Exemplos desta categoria são processos físicos que podem ser descritos por um sistema de equações diferenciais. Freqüentemente, a solução numérica de tais equações é considerada como simulação. Observe que os estados mudam continuamente e o tempo pode ser um parâmetro contínuo ou discreto. Mesmo considerando uma descrição breve das técnicas de avaliação, será dado maior foco à simulação de evento discreto.
- Simulação de eventos discretos: é a mais conhecida. Os modelos que lidam com modificações de variáveis (estados) dependentes de forma discreta em pontos específicos de tempo simulado são conhecidos como Modelos de Mudança Discreta, como mostra a Figura 2.1b. Também neste caso o parâmetro tempo pode assumir valores contínuos ou discretos.

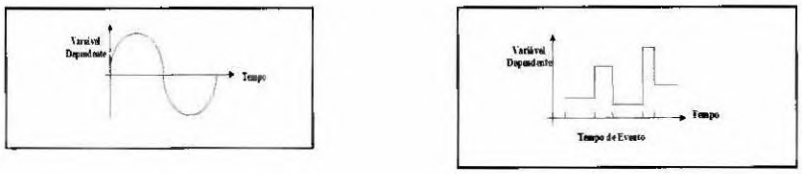

Figura 2.1 – (a) Modelagem com Mudança Contínua de Estados; (b) Modelagem com Mudança Discreta de Estados.

Num sistema de eventos discretos, o estado é modificado ao longo do tempo. A ocorrência desta mudança é causada por um *evento*. Como estão sendo consideradas mudanças de estado que ocorrem numa forma discreta em relação ao tempo, os eventos ocorrem um após o outro, ou seja, a simulação estimula evento a evento. O seqüenciamento de tais eventos e os tempos em que eles ocorrem têm uma grande importância, pois são esses valores que descrevem o desempenho de um sistema sendo simulado. O que ocorre de fato numa simulação é a *imitação* destes eventos ou seja cada evento do sistema é estimulado para que ocorra mudança de estados. Com isso, é possível obter informações como tempo entre eventos, tempo médio entre pares específicos de eventos, entre outras. Essas informações são a base para calcular as medidas de desempenho. Como a manipulação de eventos é conduzida um a um, a simulação de eventos discretos é mais fácil de tratar do que a simulação de eventos contínuos.

Então, quando se trata de simulação, pode-se entender que é um processo computacional de implementação de um modelo. Esta implementação se refere à imitação de algum fenômeno ou de um sistema dinâmico cujos estados são modificados em função de tempo. Como já dito em [Francês, 1998], John von Neuman já imaginava como a computação poderia ser aplicada na automação de processos e na modelagem de fenômenos, como, por exemplo, o famoso processo Monte Carlo, onde a distribuição de probabilidades é utilizada para descrever o comportamento de um sistema. Atualmente, a simulação é aplicada em diversas áreas graças aos avanços na área de computação, em especial no desenvolvimento de hardwares mais potentes com menor custo, bem como aos avanços em linguagens de programação. Esses avanços reduziram drasticamente os esforços consideráveis para lidar com programação de modelos de simulação.

A simulação de eventos discretos funciona orientada a tempo (simulação síncrona) ou a eventos (simulação assíncrona). Na simulação síncrona, o controle está relacionado à evolução do tempo em passos ou intervalos constantes. Então, em cada intervalo $[t, t+\Delta t]$, verifica-se se os eventos estão habilitados para que ocorra uma reação, realizando uma transição de um estado para outro. Assume-se que a ordem de disparo dos eventos não é relevante, e que eles são independentes. A grande vantagem da simulação síncrona é a facilidade de sua implementação. Por outro lado, o fato de assumir que a ordem em que os eventos ocorrem não é importante e que os eventos são independentes necessita que o Δt seja pequeno. A razão disso é evitar ou minimizar a ocorrência de eventos mutuamente dependentes. A atribuição de um valor pequeno para Δt torna a simulação muito ineficiente, já que pode não haver nenhum evento habilitado num intervalo tão pequeno. Tal fator inviabiliza o emprego de simulação síncrona.

No caso de simulação assíncrona (orientada a eventos), os incrementos de tempo variam, mas o número de eventos é constante e sempre igual a um. Isso resolve o problema de eventos mutuamente dependentes, já que somente um evento ocorre dentro de um intervalo de tempo. Neste tipo de simulação, o fato de ocorrer um evento causa a ocorrência de outros no futuro. Isso pode ser ilustrado pelo exemplo da chegada de um cliente ao banco. No caso, o cliente pode ocupar uma fila e, quando chegar a sua vez, ser atendido. O que ocorre nesta técnica é a organização dos eventos a serem habilitados. Naturalmente, o primeiro evento da lista seria o

próximo evento a ocorrer e o tempo quando deve ocorrer. E esta lista segue com os eventos futuros, obviamente, numa maneira ordenada. Na execução da simulação, o primeiro evento da lista é estimulado, observando que novos eventos poderão ser criados, devendo ser inseridos em lugares apropriados na lista de eventos [Haverkort, 1998]. É preciso enfatizar que a maioria das simulações a eventos discretos na avaliação de desempenho de áreas de computação e comuicação são do tipo baseado em eventos. A restrição aqui é a necessidade de computar os instantes de tempo nos quais os futuros eventos ocorrerão. O problema é que nem sempre isso é possível, como, por exemplo, quando distribuições de tempo muito complexas são empregadas ou até mesmo quando as variáveis são contínuas. Nesses casos, deve-se recorrer às simulações baseadas em tempo.

Ao implementar um programa para simular um modelo, o primeiro elemento que ele deve conter é o relógio onde se armazena o tempo simulado. Deve também conter uma lista consistindo de eventos que ocorrerão no futuro. Esses eventos são organizados em ordem crescente do tempo de ocorrência, levando, conseqüentemente, à sua execução na ordem correta.

A realização de simulação pode ser organizada de acordo com os seguintes itens.

- O primeiro passo para realizar uma simulação corresponde à compreensão do sistema. A definição do sistema deve ser precisa e seus objetivos, claros, para não ocorrer insucesso;
- A próxima etapa consiste em construir um modelo, escolhendo-se a técnica mais adequada para isso. A Seção 3 discute técnicas de Redes de Filas, Redes de Petri e Statecharts, sugestões que podem ser aplicadas na modelagem ou na abstração de um sistema;
- Uma vez modelado o sistema (utilizando uma das técnicas de modelagem), o sistema que deve ser avaliado. É necessário criar as entradas para o modelo, as quais podem ser hipotéticas ou baseadas a partir de alguma pré-análise. Os dados podem ser classificados como Determinísticos ou Estocásticos. Os dados determinísticos correspondem aos casos onde a seqüência de passos é conhecida. No caso estocástico, os passos que o modelo segue não são conhecidos, e sim probabilísticos, ou seja, aleatórios. Naturalmente, dependendo do resultado obtido, tais dados podem ser modificados para se ter outra avaliação do sistema, o que, aliás, constitui o intuito da avaliação de desempenho. Os resultados baseados nas entradas fornecidas são avaliados e as entradas são modificadas para melhorar o desempenho do sistema;
- O próximo passo é converter ou traduzir o modelo para que seja tratado computacionalmente. Nesse aspecto, pode-se contar com pacotes de software ou linguagens de programação específicos para este fim;
- Nesta etapa, realiza-se a verificação para garantir se o modelo traduzido funciona de acordo com o esperado. Há também uma validação para garantir se o sistema foi representado corretamente. Nesta fase, pode-se inferir a eficácia de todo o processo;
- Existe ainda outra etapa fundamental: a de documentação, para relatar claramente como foi desenvolvido o processo de simulação. Todas as fases mencionadas nos itens anteriores são discutidas, enfatizando as características principais de cada fase e os seus resultados.

Note que as fases apresentadas correspondem a uma simulação de aspecto sequencial. Mas não há nenhum impedimento em se realizar interações entre as diversas etapas. Na realidade, a grande maioria dos processos de simulação são realizados de maneira não-seqüencial, possibilitando o aproveitamento da computação paralela.

Simulação é a técnica com a qual se pode obter mais precisão dos parâmetros do comportamento do sistema. É mais flexível que a modelagem analítica, que apresenta limitações sobre as características a serem

representadas, e consegue representar qualquer nível de detalhes, sendo indicada quando os modelos analíticos não são tratáveis.

2.2.2 Abordagem Analítica

Como é possível concluir na seção anterior, a simulação para avaliação de desempenho é a aplicação mais popular, já que pode ser aplicada a todos os problemas. No entanto, essa técnica apresenta uma grande desvantagem: o tempo gasto para gerar os resultados com precisão alta.

Uma alternativa, levando em consideração o fator custo, é o uso da abordagem analítica, muito conveniente por permitir uma visão resumida do comportamento do sistema (em avaliação) rapidamente a um custo pequeno. A vantagem é que diversas alternativas de projeto podem ser exploradas. No entanto, ao aproximar um modelo, os resultados podem não ser precisos, obrigando a que se recorra à simulação, que é cara. Independentemente da técnica utilizada, é importante ressaltar que as informações dos parâmetros de entrada devem ter boa qualidade para que a avaliação de desempenho seja precisa.

Cadeias de Markov a tempo contínuo são frequentemente utilizadas para modelar confiabilidade e desempenho de sistemas nesta abordagem. A teoria de cadeias de Markov (a tempo discreto ou a tempo contínuo) permite modelar sistemas que evoluem no tempo. A base dos conceitos nessas cadeias são os estados e as transições entre eles a partir de alguma perturbação conhecida como evento. A propriedade mais importante na teoria das cadeias de Markov é que basta conhecer o estado atual para prever o comportamento estocástico futuro, sem se preocupar com o passado. Para que esta propriedade seja obedecida, os eventos que realizam as transições entre estados devem, obrigatoriamente, seguir distribuições exponenciais negativas [Tijms, 2003]. Um exemplo de cadeia de Markov está ilustrado na Figura 2.2.

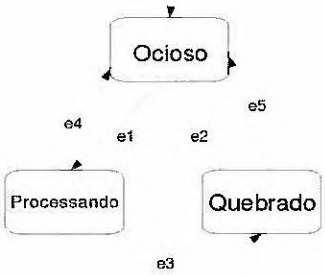

Figura 2.2 – Exemplo de uma Cadeia de Markov a tempo contínuo.

A cadeia do exemplo consiste de 3 estados (*Ocioso*, *Processando* e *Quebrado*) e 4 eventos (*e1*, *e2*, *e3*, e *e4*), que ao serem estimulados, fazem com que ocorra transição entre estados. Tais eventos seguem uma distribuição exponencial para que o diagrama seja considerado uma cadeia de Markov. Considere o exemplo da Figura 2.2 como uma cadeia de Markov representando o comportamento de um equipamento. O evento *e1* representa a chegada de uma peça para que o equipamento saia do estado *Ocioso* para *Processando*. O término do processamento é representado pelo evento *e2*, e o equipamento volta ao estado *Ocioso*. Quando o equipamento estiver no estado *Processando*, ele pode quebrar (ocorrência representada pelo evento *e3*), levando o equipamento ao estado *Quebrado*. Supondo que o equipamento seja consertado, o evento *e4* (término do reparo) leva ao estado *Ocioso*. Tais eventos devem ser associados a valores que obedeçam a uma distribuição exponencial. Por exemplo, *e1*, que representa a chegada de uma peça, pode ser associado a um

valor $\lambda 1$, descrevendo que chegam $\lambda 1$ peças, em média, numa unidade de tempo. Esses valores são conhecidos como taxas de transição. Observe a seta no estado *Ocioso*. Esta indicação é necessária para ilustrar que *Ocioso* é o estado inicial antes de os eventos começarem a ser estimulados para que se possa observar o comportamento do sistema. Note que a resolução da cadeia de Markov é possível através de métodos diretos e iterativos. Dentre os métodos diretos, são exemplos a eliminação de Gauss e o algoritmo de Grassman. Como exemplos de métodos iterativos, podemos citar os de potência, de Jacob e de Gauss-Seidel e sobre relaxação sucessiva [Bolch et al., 1998]. Uma vez modelado o sistema como uma cadeia de Markov, esta deve ser resolvida para se obterem parâmetros que possibilitem a avaliação de desempenho. Ao resolver uma cadeia de Markov, são obtidas probabilidades limite, que representam a porcentagem de tempo ocupada por cada estado num tempo infinito, e que já fornecem uma boa base para a tarefa de avaliar desempenho do sistema. Outras medidas podem ser calculadas a partir destas probabilidades. Por exemplo, resolvendo a cadeia da Figura 2.2, suponha que as probabilidades limite obtidas para os estados *Ocioso*, *Processando* e *Quebrado* sejam 0.3, 0.6 e 0.1, respectivamente. Se a equipe que projetou este equipamento achar que esses valores são razoáveis, pode encaminhar o projeto para que o equipamento seja de fato construído, para ser lançado no mercado ou com outra finalidade. No caso de a equipe achar que, por exemplo, o fato de o estado *Ocioso* estar ocupado 30% do tempo (probabilidade limite de 0.3) não seja adequado, pode modificar os valores dos eventos. Nesse caso, é possível aumentar a taxa (de transição) associada ao evento *e1* para indicar que mais peças podem ser processadas. Com isso, a ociosidade deve ser diminuída e, conseqüentemente, o estado *Processando*, ocupado por mais do que 60% do tempo (probabilidade limite de 0.6). É claro que, ao modificar os valores das taxas de transição, a cadeia deve ser resolvida novamente, para que a equipe possa conferir se os parâmetros obtidos estão satisfatórios.

2.2.3 Modelagem

A modelagem permite uma melhor compreensão do comportamento de sistemas, possibilitando uma estimativa e até mesmo uma melhoria de seu desempenho. Modelar significa que é preciso ter um meio de representar o sistema, como, por exemplo, através da utilização de uma linguagem natural, como o Português. Mas como aplicar um tratamento computacional a este modelo representado por uma linguagem natural? Com esse tipo de modelagem, só é possível compreender o comportamento de um sistema fazendo as contas utilizando uma calculadora. E quando o sistema é complexo? Será que é possível gerar as medidas a partir de uma calculadora? Para contornar isto, é necessário que a representação de um sistema seja por uma modelagem formal. Pode até ser textual. Mas neste caso, tem que ter comandos ou palavras-chave finitos para que possa ser compreendida por um programa de computador. No entanto, se houver uma possibilidade de expressar o comportamento de um sistema numa forma gráfica, a equipe que está projetando o sistema poderá contar com grande facilidade na tarefa de modelagem.

Sistemas cujo desempenho deve ser avaliado usualmente pertencem à categoria de sistemas reativos. Sistemas reativos são aqueles que respondem aos estímulos, também conhecidos como eventos. Esse tipo de sistema pode ser encontrado em uma grande variedade de dispositivos. Considere o exemplo de um forno de microondas. Seu *estado* inicial é *Parado*. Somente ao se colocar um prato e teclar alguns botões, ele começa a funcionar. A ação de teclar botões é um estímulo para que o aparelho comece a funcionar. Ao realizar esse estímulo, o aparelho sai do estado *Parado* e passa ao estado *Esquentando*, *Degelando*, ou outra função, de acordo com os botões que foram teclados.

Uma técnica muito conhecida é o Diagrama de Estados e Transições, formalizado pelas Máquinas de Estados Finitos (MEF). Um exemplo de tal diagrama já foi utilizado na Figura 2.2 para descrever uma cadeia de Markov. Isso significa que, se um sistema pode ser modelado como um diagrama de estados, este diagrama pode ser considerado uma cadeia de Markov, desde que os eventos que fazem as transições sigam uma

distribuição exponencial. Se os eventos seguirem outra distribuição, poderá ser utilizada a técnica de simulação para gerar as medidas de desempenho. Então, um diagrama de estados e transições pode ser uma excelente técnica para representar sistemas reativos, e, se for utilizada uma abordagem analítica para gerar medidas de desempenho, já estará pronta a modelagem do sistema como uma cadeia de Markov.

Para sistemas reativos constituídos por poucos estados, os diagramas de estados e transições são uma excelente opção para modelagem. O problema de representação começa a se complicar quando o número de estados é grande; nesse caso, não é possível visualizar claramente a lógica do sistema modelado imagine o núme ro de arcos de transição entre os vários estados. Para organizar uma grande quantidade de estados, seria necessária a representação de alguma forma de hierarquia. Outro problema ocorre quando o sistema a ser modelado considera atividades paralelas, o que não se pode representar explicitamente de forma direta. Todas essas limitações requerem a investigação de outras técnicas, capazes atender às demandas exígidas para representar sistemas reativos modernos.

2.2.3.1 Redes de Filas

Em uma variedade de aplicações, existem situações em que ocorre concorrência na utilização de um determinado recurso do sistema. Nessas situações, ocorre, naturalmente, um enfileiramento das requisições à espera de um determinado recurso. A modelagem que descreve a ocorrência de filas vem da teoria de filas conhecida como Redes de Filas. Os modelos de redes de filas servem para analisar a avaliação de desempenho. Uma rede de filas consiste em uma coleção de servidores, com clientes se movimentando entre eles e solicitando requisições por serviços [Tijms, 2003]. Sob algumas condições, essas redes podem ser associadas às cadeias de Markov a tempo contínuo para obtenção de parâmetros de avaliação de desempenho.

A Figura 2.3 mostra o exemplo de uma rede de filas, representando o atendimento a um serviço com apenas um servidor. Isto pode ser associado a um banco, um supermercado ou a chegada de *jobs* num processador. Em quaisquer destes exemplos, os clientes (ou *jobs*) chegam para receber atendimento para um determinado serviço e, caso o servidor esteja ocupado, formam uma fila.

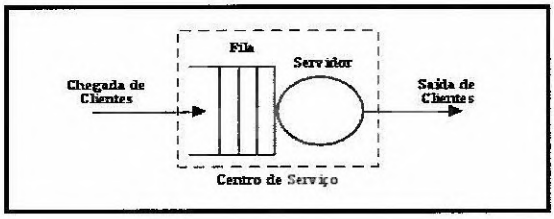

Figura 2.3 – Um exemplo de Rede de Filas [Francês, 1998].

Em [Francês, 1998], é definida uma notação para identificar sistemas de filas que consiste de seis parâmetros: a distribuição dos tempos entre chegadas (A), a distribuição do tempo de serviço (B), o número de servidores (c), a capacidade do sistema (K), o número de clientes na fonte (m) e a disciplina de fila (Z). Os parâmetros são descritos na forma $A/B/c/K/m/Z$. As distribuições dos tempos para os símbolos A e B podem ser:

(i) **G** = distribuição de tempo entre chegadas ou tempo de serviço (geral);

(ii) H = distribuição de tempo de serviço ou de tempo entre chegadas (hiperexponencial);

(iii) E = distribuição de tempo de serviço ou de tempo entre chegadas (Erlang);

(iv) M = distribuição de tempo de serviço ou de tempo entre chegadas (exponencial);

(v) D = distribuição de tempo de serviço ou de tempo entre chegadas (determinístico).

As distribuições de tempo são, de fato, distribuições de probabilidades. As mais utilizadas são: Uniforme (distribuição contínua com valores equiprováveis definidos entre um limite mínimo e um máximo), Exponencial (distribuição segundo a qual a probabilidade de acontecer um evento em um pequeno intervalo de tempo é proporcional ao tamanho desse intervalo), Hiperexponencial (caso particular de distribuição exponencial que admite uma variância muito grande em relação à média), de Erlang (distribuição derivada da soma de um número inteiro de variáveis aleatórias independentes e exponencialmente distribuídas) e de Poisson (distribuição discreta que estabelece que a duração do intervalo de tempo entre chegadas é exponencialmente distribuído e as chegadas ocorrem uma de cada vez). O atendimento ao cliente pode usar uma das seguintes disciplinas: FCFS (*First Come First Served*), LCFS (*Last Come First Served*), RR (*Round Robin*). Cada uma delas pode ou não atribuir prioridades aos clientes.

A notação *A/B/c/K/m/Z* usualmente empregada pode ser simplificada como *A/B/c* quando não há limite para o tamanho da fila, a fonte de clientes é infinita e a disciplina da fila é FCFS. Assim, um sistema de fila *M/M/1* é caracterizado por apresentar taxas de chegadas entre clientes e tempos de serviço que obedeçam à distribuição exponencial e possuir um único servidor. Nesse caso, algumas fórmulas podem ser aplicadas para obter medidas de desempenho. Considerando λ como a taxa de chegada e μ como a taxa de serviço, o número médio de clientes numa fila é dado por $(\lambda^2)/\mu(\mu-\lambda)$; o número médio de clientes no sistema é dado por $\lambda/(\mu-\lambda)$; o tempo médio por cliente na fila corresponde a $(\lambda)/\mu(\mu-\lambda)$; o tempo médio por cliente que permanece no sistema é igual a $1/(\mu-\lambda)$; e a intensidade de tráfego corresponde a λ/μ.

2.2.3.2 Redes de Petri

Redes de Petri são capazes de especificar sistemas através de uma representação matemática. Estas redes possuem mecanismos de análise poderosos, que permitem a verificação de propriedades e da correção do sistema especificado [Maciel et. al., 1996]. As redes de Petri permitem representar paralelismo e concorrência, levando em consideração sincronização e não-determinismo, que são características essenciais em sistemas reativos modernos. Para representar graficamente um sistema reativo em uma rede de Petri, são necessários dois elementos: lugar (representado por um círculo) e transição (representado por uma barra), como mostra a Figura 2.4.

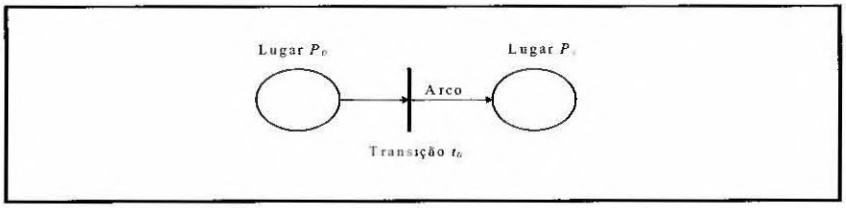

Figura 2.4 – Elementos básicos de uma rede de Petri [Francês, 1998].

Os lugares correspondem aos estados que o sistema pode ocupar, enquanto as transições correspondem a respostas aos estímulos, que representam mudanças de um lugar para outro. A conexão entre os dois elementos é realizada através de arcos direcionados. As redes de Petri trabalham com marcas (*tokens*), que fazem parte dos lugares. A marcação é utilizada para observar o comportamento dinâmico da rede. As redes evoluíram para poderem lidar com diferentes aspectos na representação de sistemas reativos, gerando redes de Petri coloridas, hierárquicas, temporais, determinísticas, estocásticas e estocásticas generalizadas. As últimas duas são bastante utilizadas para lidar com avaliação de desempenho.

Uma vez representado o modelo de um sistema como uma rede de Petri, a solução para obtenção das medidas de desempenho pode ser obtida através da simulação ou da abordagem analítica. No segundo caso, essa representação deve ser convertida para uma cadeia de Markov, e as probabilidades limite devem ser calculadas [Bolch et al., 1998].

2.2.3.3 Statecharts

Os Statecharts foram criados para especificar e simular sistemas de tempo real [Harel, 1987]. Eles são de fato diagramas de estado, mas possibilitam representar hierarquia e atividades paralelas. Eles ficaram tão populares que foram adotados para representar o comportamento de métodos orientados a objetos em UML [Booch et al., 1999]. Os elementos básicos são estados (representados por retângulos arredondados), transições entre estados (arcos direcionados), eventos que são responsáveis para que haja transições, condições de guarda e ações (eventos que afetam outras transições).

Estados são agrupados para representar profundidade (hierarquia). Com isto, é possível combinar um conjunto de estados com transições comuns em um super-estado. Os super-estados podem ser refinados em sub-estados (abordagem *top-down*). O refinamento pode ser um XOR (ou exclusivo) ou AND. Quando um super-estado XOR estiver ativo, somente um de seus sub-estados poderá estar ativo (não mais de um). No caso do super-estado AND estar ativo, todos os seus sub-estados estarão ativos. Os estados que não possuem sub-estados, ou seja, não decompostos, são conhecidos como *estados básicos*.

Por definição, é necessário sempre definir um estado inicial, que é identificado por uma seta. Não se deve esquecer que, no caso do super-estado AND, haverá mais de um estado inicial. Por isso, nos Statecharts, o estado global é conhecido como configuração, onde uma configuração inicial representa estados básicos iniciais de cada componente paralelo. No entanto, pode-se ignorar o estado inicial quando há necessidade de "lembrar" o último estado ativo. Para isto, os Statecharts tem o símbolo H para não respeitar mais o estado inicial. O símbolo H só atinge o primeiro nível de hierarquia e, quando há interesse em que todos os níveis de profundidade sejam aplicados, utiliza-se o símbolo H*.

Os eventos responsáveis pela ocorrência de transições entre estados são divididos em duas categorias: externos e internos. Os externos são aqueles explicitamente estimulados por uma entidade ou observador externo. Os internos não necessitam ser explicitamente estimulados, pois são detectados pelos Statecharts, e a reação ocorre imediatamente (por essa razão, tais eventos são também conhecidos como eventos imediatos). Mesmo que um evento possa ser estimulado, é possível "guardá-lo" ou "protegê-lo" associando-se uma condição a ele, através da notação *evento[condição]*, interpretada da seguinte forma: o evento pode ser habilitado somente se a condição for satisfeita, permitindo, assim, a ocorrência da transição. Uma condição muito utilizada nos Statecharts é a *in[S]*, onde *S* é um estado. Esta condição se torna verdadeira quando o estado *S* se torna ativo. Uma ação também pode estar associada ao evento, estendendo a notação para *evento[condição]/ação*. Nesse caso, se o evento for acionado (supondo que a condição foi satisfeita), será disparada a transição e a reação continuará, pois a ação (um evento), em geral, afeta outro componente paralelo, levando a outra transição. Esta

é uma forma de sincronização. A ação é considerada como um evento interno, ou seja, não há necessidade de estimulá-la explicitamente. A Figura 2.5 mostra um exemplo representado em Statecharts.

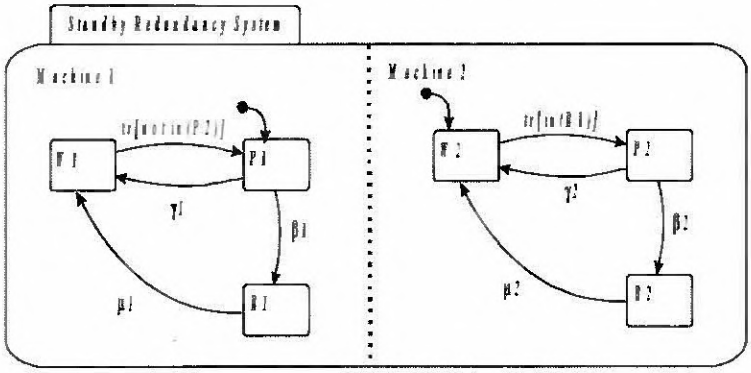

Figura 2.5 – Um Sistema de Redundância especificado em Statecharts.

2.2.3.4 PerformCharts

Ao estudar e conhecer os Statecharts, observa-se que é possível representar o comportamento de uma variedade de sistemas reativos, além de obter uma visualização clara do sistema representado. Isso se deve à riqueza dos elementos que os Statecharts possuem. Então, foi vislumbrada e desenvolvida a idéia de gerar medidas de desempenho utilizando uma abordagem analítica de sistemas reativos especificados em Statecharts [Vijaykumar et al., 2002]. Essa idéia resultou no desenvolvimento da ferramenta PerformCharts. Como já foi mencionado, para ser realizar uma avaliação de desempenho utilizando uma solução analítica, é necessário que o sistema esteja modelado como uma cadeia de Markov. Então, como se associa uma cadeia de Markov a uma especificação em Statecharts?

A solução é idêntica àquela utilizada quando um sistema é modelado como uma rede de Petri. Ou seja, a cadeia de Markov é gerada a partir da especificação Statecharts. Antes de mostrar como a especificação Statecharts é convertida, é necessário uma explicação breve sobre os eventos externos. Para que uma cadeia de Markov seja considerada, é preciso lembrar que os eventos entre transições devem estar associados às taxas de transição com distribuição exponencial negativa. Então, ao se especificar ou modelar um sistema reativo em Statecharts, assume-se que os eventos externos (também chamados de eventos estocásticos, neste caso) estão associados às taxas com distribuição exponencial negativa.

O algoritmo para converter a especificação Statecharts em uma cadeia de Markov será descrito informalmente:

1. obtém-se a configuração inicial;
2. eventos internos, se existirem, serão estimulados para gerar uma nova configuração;
3. para a configuração obtida, eventos externos (estocásticos) serão estimulados;
4. para cada evento externo estimulado, obtém-se uma nova configuração;
5. para cada nova configuração obtida, são estimulados, se existirem, eventos internos, gerando nova configuração;
6. o processo prossegue até que não haja mais configurações para as quais seja necessário estimular mais eventos.

O final deste processo gera uma estrutura formada por uma lista de configurações fonte, configurações destino, e eventos (com as taxas) que levaram à transição de cada configuração fonte para a configuração destino correspondente. Já que os eventos referentes às transições entre configurações seguem uma distribuição exponencial negativa, a estrutura é considerada uma cadeia de Markov. Quando a cadeia é resolvida, obtêm-se as probabilidades limite, base para obtenção das medidas de desempenho. Para entender melhor o algoritmo, considere a Figura 2.5, representando duas máquinas, onde uma serve como redundância para outra quando uma delas estiver quebrada. As duas máquinas são representadas como componentes paralelos (*Machine 1* e *Machine 2*), e apresentam o mesmo comportamento, ou seja, seus sub-estados são W1 e W2 (*ocioso*), P1 e P2 (*processando*), B1 e B2 (*quebrada*). O estado inicial é representado por P1 e W2, respectivamente, na primeira e na segunda máquina. Os eventos externos são: $\gamma_1, \mu_1, \beta_1, \gamma_2, \mu_2, \beta_2$. Há dois eventos internos: *tr{not in[P2]}* e *tr{in[B1]}*. O primeiro é habilitado quando um dos sub-estados do componente *Machine 2* estiver em W2 ou B2. O segundo é habilitado somente quando o sub-estado do componente *Machine 1* estiver em B1.

A configuração inicial do sistema ilustrado na Figura 2.5 é {P1, W1}. Para essa configuração, não há eventos internos que possam ser habilitados. Então, são listados quais eventos externos (estocásticos) podem ser estimulados; no caso, γ_1 e β_1. Ao se estimular γ_1, a nova configuração obtida é {W1,W2}; verifica-se agora se há algum evento interno que possa ser habilitado, obtendo, neste caso, resposta negativa. Ainda para a configuração inicial, há o evento β_1 que precisa ser estimulado, gerando a configuração {B1, W2}. Nesse caso, no componente *Machine 1*, ao se ativar o sub-estado B1, imediatamente se torna verdadeira a condição *in[B1]*, o que leva à transição do o estado W2 para o estado P2 no componente *Machine 2*. Então, a configuração {B1, W2} torna-se uma configuração intermediária, gerando diretamente a configuração {B1, P2}. Este processo continua até que a cadeia seja obtida. A cadeia de Markov para o sistema ilustrado na Figura 2.5 está apresentada na Figura 2.6.

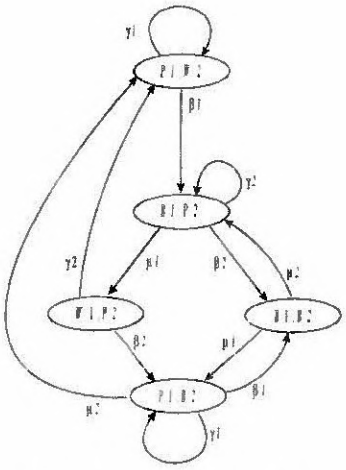

Figura 2.6 – Cadeia de Markov da Figura 2.5.

Considere os valores das taxas de transição, que são as entradas fornecidas junto com o modelo representado em Statecharts, mostradas na Tabela 2.1.

Tabela 2.1 - Valores das taxas de transição da Figura 2.6.

	{P1, W2}	{B1, P2}	{W1, P2}	{B1, B2}	{P1, B2}
{P1, W2}	γ_1 (5.0)	β_1 (0.1)	0.0	0.0	0.0
{B1, P2}	0.0	γ_2 (5.0)	μ_1 (3.0)	β_2(0.2)	0.0
{W1, P2}	γ_2(5.0)	0.0	0.0	0.0	β_2 (0.2)
{B1, B2}	0.0	μ_2 (3.0)	0.0	0.0	μ_1 (3.0)
{P1, B2}	μ_2 (3.0)	0.0	0.0	β_1 (0.1)	γ_1 (5.0)

Ao resolver a cadeia da Figura 2.6 através da execução da ferramenta PerformCharts, obtêm-se as seguintes probabilidades limite, exibidas na Tabela 2.2.

Tabela 2.2 - Probabilidades Limite da Cadeia de Markov da Figura 2.6.

Configurações	Probabilidades Limite
{P1, W2}	0.948475
{B1, P2}	0.030631
{W1, P2}	0.0176717
{B1, B2}	0.00105715
{P1, B2}	0.00216554

A ferramenta PerformCharts (escrita em linguagem C++) ainda não possui uma interface gráfica. Inicialmente, priorizou-se a necessidade de escrever um programa principal em linguagem C++ para especificar um sistema reativo em Statecharts. Neste módulo principal, também são incluídas chamadas aos métodos para gerar a cadeia de Markov e sua resolução. Para melhorar a interface a curto prazo, foi proposta a linguagem PcML (PerformCharts Markup Language) [Amaral et al., 2004], baseada em XML, onde somente a especificação do sistema em Statecharts seria suficiente. Programas em Perl e Java foram desenvolvidos para interpretar a interface textual (em PcML) e gerar o módulo principal em C++. Na configuração atual da ferramenta, basta conectar este módulo ao restante das classes, gerando um arquivo executável que, ao ser executado, gera como saída as probabilidades limite.

2.3 Considerações Finais

Em se tratando de sistemas complexos com vários componentes paralelos, cada um com centenas de milhares de estados, a modelagem de sistemas reativos não é uma tarefa simples. Descrever tal sistema como uma cadeia de Markov é uma tarefa árdua, o que justifica o número de pesquisas na área de modelagem. É difícil afirmar que uma técnica seja melhor que outra. Mesmo existindo fortes argumentos, deve-se considerar a influência dos pesquisadores acostumados a uma determinada técnica.

O texto descrito na parte A do documento sugere o uso dos Statecharts, já que eles são derivados de diagramas de estados e transições naturais para descrever o comportamento de sistemas reativos. A vantagem dos Statecharts é a maneira explícita de representar hierarquia e paralelismo. Além disso, foi mostrado que é possível associar uma solução (cadeia de Markov) à sua especificação.

A previsão do comportamento de alguns sistemas necessita que haja especificação de transições probabilísticas. Este aspecto faz parte da ferramenta PerformCharts, na qual também já está implementada a aplicação do histórico (H e H*). Há trabalhos em andamento cujo objetivo é possibilitar a geração da cadeia de Markov no formato XML, o que pode ser uma tarefa interessante. Em breve, espera-se que a ferramenta esteja disponível para utilização via Internet. O desenvolvimento de uma interface gráfica, que deve facilitar a especificação, é outro trabalho em desenvolvimento.

Referências

Amaral, A.S.M.S.; Veloso, R.R.; Vijaykumar, N.L.; Francês, C.R.L.; Oliveira, E. On proposing a Markup Language for Statecharts to be used in Performance Evaluation. **International Journal of Computational Intelligence**. Vol. 13(1), pp. 260-265, 2004

Bolch, G.; Greiner, S.; de Meer, H.; Trivedi, K.S. **Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications**. John Wiley & Sons, Inc., U.S.A., 1998

Booch, G.; Rumbaugh, J.; Jacobson, I. **The Unified Modeling Language User Guide**. Addison-Wesley Longman, Inc., U.S.A., 1999

Francês, C.R.L. **Stochastic Feature Charts: Uma Extensão Estocástica para os Statecharts**. Dissertação de Mestrado em Ciência da Computação. ICMC/USP São Carlos, 1998

Harel, D. A Visual Formalism for Complex Systems. **Science of Computer Programming**. Vol. 8, pp. 231-274, 1987

Haverkort, B.R.; **Performance of Computer Communication Systems: A Model-Based Approach**. John Wiley & Sons, Ltd., U.S.A., 1998

Maciel, P.R.M.; Lins, R.D.; Cunha, P.R.F. **Introdução às Redes de Petri e Aplicações**. Escola de Computação da SBC, Campinas, 1996

Soares, L.F.G.; **Modelagem e Simulação Discreta de Sistemas**. Editora Campus Ltda., Brasil, 1992

Tijms, H.C. **A First Course in Stochastic Models**. John Wiley & Sons, Ltd., England, 2003

Vijaykumar, N.L.; Carvalho, S.V.; Abdurahiman, V. On proposing Statecharts to specify Performance Models. **International Transactions in Operational Research (ITOR)**. Vol. 9(3), pp.321-336, 2002

Parte B: Modelagem e Geração Automática de Seqüências de Testes (em colaboração com Valdivino Alexandre de Santiago Júnior, INPE, São José dos Campos, SP)

2.4 Introdução

Atualmente, depois de uma diminuição drástica dos custos de sistemas computacionais, eles têm sido utilizados mais intensamente em áreas diversas e, muitas vezes, críticas, tais como: aplicações espaciais, controle de tráfego aéreo, tratamentos médicos, indústria automotiva, redes de comunicação e outras.

Esse avanço tecnológico exige que sistemas de software e hardware sejam cada vez mais complexos, não somente em termos de tamanho, mas também de funcionalidade. A demanda de complexidade maior nos sistemas atuais conseqüentemente leva à produção de sistemas menos confiáveis, se não for aplicada alguma metodologia no seu desenvolvimento. A presença de erros nesses sistemas pode trazer conseqüências catastróficas para projetos que utilizam uma quantidade enorme de recursos financeiros e tempo e, dependendo

da aplicação, pode trazer risco à população, ao meio ambiente, etc., exigindo, portanto, alto grau de confiabilidade. Dessa forma, é fundamental o uso de metodologias com disciplina, que garantam a qualidade desde o início e durante todo o processo de desenvolvimento do sistema, para que o produto final saia com alto grau de confiabilidade [Amaral, 2005].

Institutos ou organizações que desenvolvem software devem entender que uma das mais importantes técnicas para gerar software de qualidade é a realização de testes. Na verdade, testes devem ser realizados em todas as fases do ciclo de desenvolvimento de um software: especificação, projeto e codificação [Pressman, 2004]. Testes têm um custo muito alto e, por isso, devem ser bem planejados. O INPE (Instituto Nacional de Pesquisas Espaciais), por exemplo, é responsável pela construção de satélites que carregam a bordo instrumentos científicos complexos e caros, além de computadores. O software embutido em tais computadores é complexo, pois interage com hardware do computador, sensores, atuadores e outros dispositivos presentes no satélite. No caso de falhas, não há possibilidade de substituição quando se trata de missões não tripuladas [Santiago et al., 2006]. Então, é necessário contar com as atividades de verificação e validação durante todo o processo do ciclo de desenvolvimento.

O objetivo da parte B deste documento é mostrar numa forma bem resumida a importância de atividades de testes nas áreas de verificação e validação de software. A Seção 2 descreve alguns conceitos básicos sobre testes. Ainda nesta Seção, serão explicados alguns métodos de geração de testes e a modelagem utilizada para esta geração. A Seção 3 descreve como a ferramenta PerformCharts (inicialmente desenvolvida para avaliar desempenho de sistemas reativos) foi adaptada para ser utilizada na geração de casos de testes de sistemas especificados em Statecharts. Algumas considerações serão comentadas na Seção 4.

2.5 Testes

Na década de 1970, 50% de tempo e 50% do custo eram gastos com testes para garantir a confiabilidade do software desenvolvido. Atualmente, as linguagens de programação evoluíram nos conceitos de projeto e desenvolvimento. Além disso, há ambientes de desenvolvimento para conduzir o desenvolvedor a gerar código com maior confiabilidade. Mesmo assim, as atividades de teste continuam ocupando 50% de tempo e 50% dos recursos do projeto [Myers, 2004]. Há casos de acidentes que ocorreram com equipamentos médicos, aviões e foguetes devido ao mau planejamento e projeto de testes. O efeito colateral desses erros foi a perda de recursos financeiros e, em cenários ainda piores, a perda de vidas humanas.

Há várias definições para os termos *verificação* e *validação*. Uma das mais utilizadas pode ser encontrada em [IEEE, 1998] e é descrita a seguir. A verificação é o processo no qual deve haver provas objetivas para assegurar que no software está implementada corretamente uma função específica (ou seja, *o produto foi construído corretamente?*). A validação é o processo no qual deve haver provas objetivas para assegurar que o produto está em conformidade com os requisitos do software, ou seja, que o produto certo está sendo desenvolvido (ou seja, *o produto certo foi construído?*). Os dois processos são popularmente conhecidos como atividades V & V, e as provas objetivas para assegurar que essas duas atividades foram alcançadas são realizadas através da aplicação de testes.

Teste é uma atividade que tem um único objetivo: encontrar erros. Um bom teste é aquele que encontra o maior número de erros num código [Myers, 2004]. Testes devem garantir que o produto realize as tarefas para as quais foi projetado e nada além disso. Para garantir que os testes sejam de qualidade, deve haver uma estratégia para planejar e projetar casos teste, executá-los e depois analisar os resultados para possibilitar um veredito sobre o produto. A maioria das estratégias sugere que os testes devem ser realizados começando com nível de unidade (menor componente de um software – *função*, *subrotina*, *método*) e depois expandindo os níveis (integrando as unidades) até chegar ao sistema como um todo.

Teste de unidade é fundamental, e, num bom projeto, a unidade deve ser pequena, possibilitandoa geração de casos de teste rapidamente; se erros forem encontrados, a implementação da correção deve ser fácil. Quando se testam unidades, é comum encontrar chamadas a outras unidades e, neste caso, são desenvolvidos *stubs*, para substituir os módulos que estão sendo chamados. É também necessário desenvolver *drivers* que são, na verdade, programas principais que chamam a unidade em teste. Se esses testes não forem conduzidos numa forma sistemática, o impacto de erros não encontrados pode ser muito grande no momento da integração das unidades e dos testes sobre tal integração, ocasionando uma reação em cadeia que afeta até o teste do sistema.

Testes de integração correspondem à junção das unidades que acabaram de ser testadas. Nesse tipo de teste, não há mais *stubs* ou *drivers*, e o objetivo é encontrar erros de interface entre as unidades e de agrupamento das unidades. Não se trata de um teste simples, pois há várias formas de combinar as unidades. Por exemplo: será que as unidades A e B são combinadas para testar a sua integração ou seria melhor combinar unidades A e C?

Uma vez terminada a fase de teste de integração, começa o teste de sistema, o que significa que o sistema de software já está montado. A preparação para esta fase deve começar durante a especificação dos requisitos, quando é necessário planejar claramente os testes, além de gerar testes baseados na especificação (também conhecidos como *testes caixa preta*) [Burnstein, 2003]. No teste de sistema, deve-se considerar o seguinte:

- o sistema deve funcionar de acordo com os requisitos;
- o comportamento funcional deve ser avaliado;
- requisitos de qualidade como confiabilidade, usabilidade, desempenho e segurança devem ser avaliados;
- defeitos nas interfaces externas como *deadlocks*, interrupções e uso ineficiente de memória devem ser detectados;
- a qualidade do sistema deve ser bem medida antes de convidar os clientes;
- testes devem ser realizados por uma equipe independente na qual não deve haver membros que participaram do desenvolvimento.

O teste de sistema engloba os seguintes testes:

- Teste Funcional
  - Também conhecido como *testes caixa preta*. Entradas são exercitadas e as saídas são observadas e comparadas com as saídas esperadas.
- Teste de Desempenho
  - É um objetivo não-funcional. É possível detectar fatores de hardware/software que impactam no desempenho. Isso deve permitir uma otimização na alocação de recursos, por exemplo.
- Teste de Stress
  - O sistema deve ser testado a um volume de dados no seu pico, ou seja, os recursos foram alocados na faixa máxima.
- Teste de Configuração
  - São verificados os efeitos no sistema quando há mudanças de hardware, por exemplo.
- Teste de Segurança
  - Este teste corresponde a *Safety* e *Security*. O objetivo é garantir a integridade e a confidencialidade

dos dados.

- Teste de Recuperação
  - Este teste é importante para mostrar que o sistema continua seguro mesmo perdendo um recurso.
- Teste de Regressão
  - Quando o software for modificado, os testes que já passaram devem ser aplicados de novo.
- Teste de Confiabilidade
  - É um assunto muito complexo e está sendo muito discutido quando se trata de tecnologias de acesso como WLAN, PLC, etc.
- Teste de Usabilidade
  - Devem ser considerados fatores humanos como idade, cultura, etc., por exemplo, na interface com o sistema.

Métodos de Geração de Casos de Teste baseados em Modelos

Como já foi mencionado, o objetivo do teste é executar um programa e achar erros. Um bom caso de teste é aquele que tem alta probabilidade de detectar um erro ainda não descoberto; um caso de teste é bem sucedido quando encontra um erro ainda não descoberto. Além disso, o objetivo é descobrir o maior número possível de erros deve ser em menor tempo e esforço possíveis [Pressman, 2004]. Até agora, falou-se de casos de teste; mas o que seriam esses casos? Para poder realizar testes numa forma sistemática [Beizer, 1990], foram desenvolvidos métodos para geração de casos de teste. Dependendo da abordagem, tais casos podem ser gerados ainda na fase de especificação dos requisitos, e executados quando a implementação estiver pronta. A análise dos resultados obtidos fornece conclusões sobre a qualidade da implementação.

Testes podem ser projetados de duas formas:

- Testes Funcionais (*Caixa Preta*): o produto é considerado como uma *caixa preta* onde estimulam-se as entradas e verifica-se se as saídas estão de acordo com o comportamento especificado;
- Testes Estruturais (*Caixa Branca*): o produto é conhecido pela sua estrutura interna, ou seja, tem-se acesso aos detalhes da implementação, estilo, banco de dados utilizados, etc.

Neste minicurso, serão abordados testes funcionais, cuja estratégia se baseia em entradas e saídas. No entanto, não é fácil selecionar um subconjunto adequado dentro de um domínio de entradas. Em uma função de raíz quadrada, por exemplo, o correto seria estimular a função com todos os valores positivos, o que consumiria tempo infinito. E como ficam os valores negativos? E as frações? O processo de teste fica inviável. Então, a solução está na mudança de abordagem. No lugar de uma prova absoluta, deve-se visar a uma demonstração convincente [Burnstein, 2003]. Ou seja, deve haver uma mudança de abordagem, permitindo a realização de testes suficientes para garantir que a probabilidade de falha devido aos erros que hibernam é baixa o suficiente para se aceitar o produto. Agora, a pergunta: o que é suficiente? Não há uma resposta definitiva para esta pergunta. O *suficiente* depende da aplicação do produto; se a aplicação para a qual o produto foi projetado for crítica, então, deve-se dedicar um maior tempo e mais recursos para o seu teste.

As seguintes abordagens são utilizadas para gerar os casos ou seqüências de testes: particionamento por classes de equivalência, análise de valor limite, tabelas de decisão e transição de estados. O foco deste documento será a geração de casos de testes por meio de transição de estados.

Não se pode testar algo que não se entende. Deve-se ter um conhecimento sobre o que a IST (Implementação Sob Teste) faz. Uma forma de conhecer a IST é o uso da modelagem para captar as informações do seu funcionamento. Como o objetivo é gerar casos de testes funcionais, o ideal seria modelar o comportamento do software de acordo com os requisitos. A modelagem mais utilizada para representar o comportamento é a Máquina de Estados Finitos (MEF), expressa graficamente por um diagrama de estados e transições. Um estado pode corresponder a alguma função do software, que deve reagir a algum estímulo (entrada), produzindo uma saída e mudando para outro estado. Uma MEF pode ser representada na forma de um grafo, e a geração de casos de testes acaba se transformando em um problema de busca. Um exemplo de uma MEF está representado na Figura 2.7.

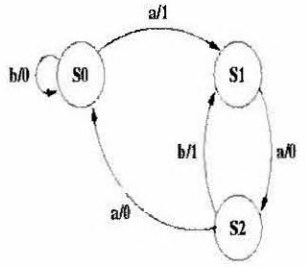

Figura 2.7 – Exemplo de uma Máquina de Estados.

Os estados são S0, S1 e S2 onde o estado S0 é o estado inicial. Os eventos (ou entradas) são a e b e as saídas são 0 e 1. No estado S0, ao estimular a entrada *b*, a MEF permance no estado S0 produzindo uma saída *0*. A partir do mesmo estado S0, ao estimular a entrada *a*, a MEF muda para o estado S1 produzindo uma saída *1*. A mesma MEF pode ser representada por uma tabela (ou matriz), como mostra a Figura 2.8.

	S0	**S1**	**S2**
S0	b/0	a/1	-
S1	-	-	a/0
S2	a/0	b/1	

Figura 2.8 – MEF da Figura 2.7 representada como uma matriz ou tabela.

A geração de casos de teste consiste em definir um conjunto de seqüências de eventos partindo de um estado inicial e voltando a ele. Para cada passo de estímulo de um evento, devem ser definidos o próximo estado, a transição e a saída, ou seja, o conjunto de testes consiste de 3 partes: seqüências de eventos (entradas), transições correspondentes (ou próximos estados) e saídas. Como é impraticável gerar casos de teste que

cubram todos os possíveis caminhos, é necessário gerar um conjunto restrito. Uma abordagem que se pode adotar é a garantia de que todos os estados foram acionados. Outra abordagem, mais prática, é a garantia de que toda as transições foram acionadas. Lembrando que a geração de casos de testes é um problema de busca, são listados a seguir alguns métodos que podem ser utilizados nesta tarefa [Sidhu & Leung, 1989].

- Método T (*Transition Tour*): A seqüência é gerada aplicando-se entradas (eventos) aleatórias até que a máquina tenha passado por toda transição pelo menos uma vez. A Figura 2.9 mostra outro exemplo de MEF, a partir da qual será mostrada uma seqüência obtida através do método *Transition Tour*. A Figura 2.10 mostra a seqüência dos eventos e os próximos estados.

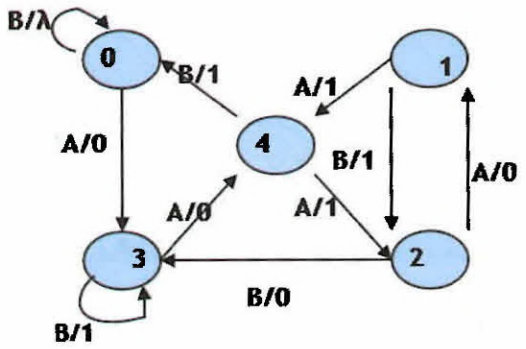

Figura 2.9 – Um outro exemplo de uma MEF [Sidhu & Leung, 1989].

B	A	B	A	B	A	A	A	A	A	A	A	B	B
0	0	3	3	4	0	3	4	2	1	4	2	1	2

Figura 2.10 – Seqüência de Testes a partir do Método Transition Tour [Sidhu & Leung, 1989].

- Método *UIO (Unique Input/Output)*: Para um dado estado de uma máquina, o comportamento de entradas e saídas não é apresentado por nenhum outro estado da máquina. Se uma seqüência de entradas for aplicada a um estado S de uma máquina, e a saída obtida for s_1, s_2, ..., é fato que a máquina estava no estado S antes de serem aplicadas as entradas. Considere a mesma máquina apresentada na Figura 2.9. Um exemplo de seqüência UIO, aplicando as entradas A, A, A e B partindo do estado 0, é mostrado na Figura 2.11.

Estado	0	3	4	2
Entrada	A	A	A	B
Saída	0	0	1	0
Próximo Estado	3	4	2	3

Figura 2.11 – Seqüência de Testes a partir do Método UIO [Sidhu & Leung, 1989].

- Método D (*Distinguishing Sequence*): Uma seqüência de entradas é uma seqüência distinta (*distinguishing sequence*) se a seqüência de saídas for diferente para cada estado inicial. Observe a Figura 2.12, ilustrando uma MEF com 3 estados; nesta ilustração, é possível observar que *ab* é uma seqüência distinta partindo do estado S1, pois, aplicando a seqüência para o estado S1, a saída obtida é 01, enquanto aplicada para os estados S2 e S3, as saídas são 11 e 00, respectivamente.

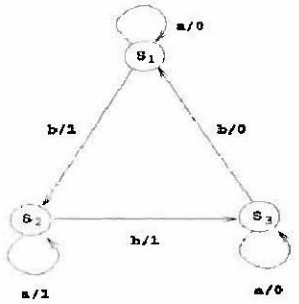

Figura 2.12 – MEF com 3 estados [Lee & Yannakakis, 1996].

- Método *Switch Cover*: Este método combina caminhos para cobrir o grafo [Beizer, 1990] e [Chow, 1978]. A especificação 0-*switch* testa cada transição individualmente. 1-*switch* testa seqüências com duas transições. E, (N-1)-*switch* é a seqüência máxima onde N é o número de estados. Um algoritmo específico, do método *switch cover*, implementado dentro de uma ferramenta, desenvolvida pela UNICAMP, CONDADO [Amaral, 2005] está mostrado na Figura 2.13, onde o item (a) mostra a MEF para a qual casos de testes devem ser gerados. O passo (b) da Figura 2.13 mostra o processo onde os eventos nos arcos (a, r, b, f, e c) são convertidos em nós. Os arcos dirigidos são criados obedecendo o seguinte: se existe um nó na Figura 2.13a onde incide um arco e de onde sai um outro, então cria um arco entre estes nós – como exemplo, foi criado um arco ligando nó a e nó b, pois na máquina original (Figura 2.13a), o arco a chega no estado P e o arco b sai deste estado P. Um terceiro passo (c) gera um grafo "Eulerizado" para manter um balanço polarizado entre os nós que chegam e saem. Quando não há este balanço, há uma duplicação de um arcos já existentes para manter o número igual entre os arcos que entram e os arcos que saem.

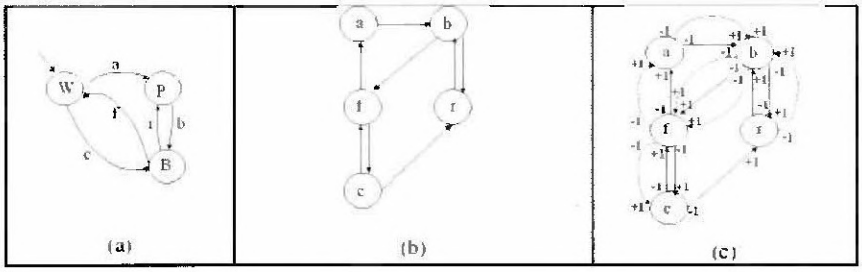

Figura 2.13 – (a) MEF; (b) Grafo dualizado com arcos; e (c) Grafo "Eulerizado" [Santiago et al., 2006]

2.6 PerformCharts

Esta ferramenta já foi vista na Parte A deste documento, onde foi explicado que ela foi desenvolvida para avaliar o desempenho de sistemas especificados em Statecharts. As medidas de desempenho podem ser obtidas após a conversão da especificação em Statecharts para uma cadeia de Markov.

Como já foi visto também, os Statecharts podem especificar sistemas com complexidade alta, pois permitem uma representação explícita de hierarquia e de atividades paralelas, o que facilita lidar com sistemas com centenas de estados e transições. Nesta aplicação, a idéia é semelhante à que ocorre com a avaliação de desempenho, no sentido de utilizar Statecharts para representar a especificação de um software (de acordo com os requisitos). A solução também é semelhante, ou seja, nos casos de teste, a especificação Statecharts é convertida para uma Máquina de Estados Finitos (MEF) ou diagrama de estados e transições (lembrando que uma cadeia de Markov é representada por um diagrama de estados e transições). A diferença é que, para uma MEF que não é considerada uma cadeia de Markov, os eventos (ou entradas) não precisam obedecer à distribuição exponencial.

Então, se a especificação de um software for representada por Statecharts e a ferramenta PerformCharts gerar a MEF, a questão passa a ser a aplicação de um ou mais métodos, dentre os já descritos, para a geração de casos de teste. Quando a ferramenta PerformCharts foi adaptada para ser utilizada na geração de casos de testes [Amaral, 2005], o método escolhido foi o *switch cover*, através da utilização da ferramenta CONDADO. Para que as duas ferramentas possam ser integradas, foi adotado o processo descrito a seguir, ilustrado na Figura 2.14.

A representação da especificação de um software é escrita na linguagem PcML, que é interpretada por um *script* codificado em Perl para gerar o programa principal na linguagem C++. Ao associar este módulo principal às demais classes e executar a ferramenta PerformCharts, é gerada a MEF em linguagem XML. Um *parser* escrito na linguagem XSLT é aplicado para transformar a MEF em uma base de fatos, que é a forma de entrada para a ferramenta CONDADO. A execução da ferramenta CONDADO resulta na geração dos casos de teste.

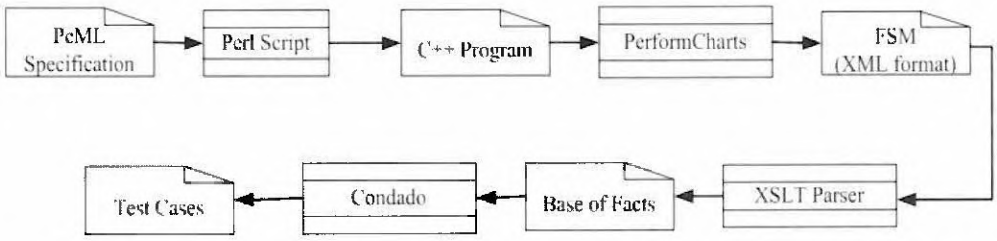

Figura 2.14 – Metodologia utilizada para integrar as ferramentas PerformCharts e CONDADO.

Para ilustrar o processo, desde a especificação até a geração dos casos de testes, considere a Figura 2.15, que representa o protocolo TCP/IP. A Figura 2.16 mostra a MEF gerada pela ferramenta PerformCharts. A Figura 2.17 mostra uma parte da base de fatos, convertida a partir da MEF, que é a entrada para a ferramenta CONDADO. A Figura 2.18 mostra uma parte dos casos de teste gerados pela ferramenta CONDADO.

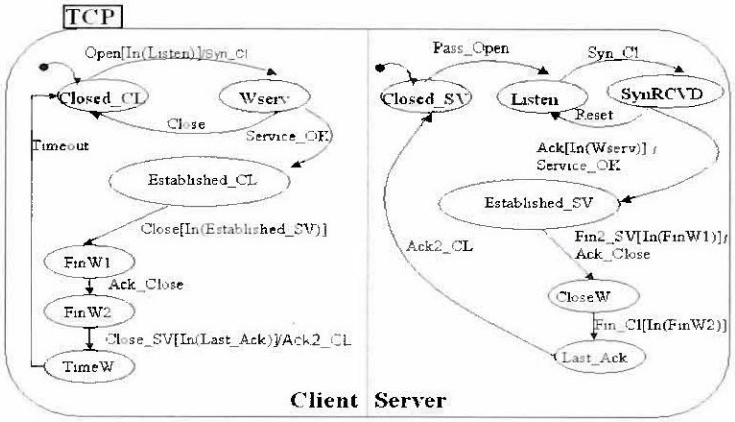

Figura 2.15 – Especificação do Protocolo TCP/IP em Statecharts [Amaral, 2005].

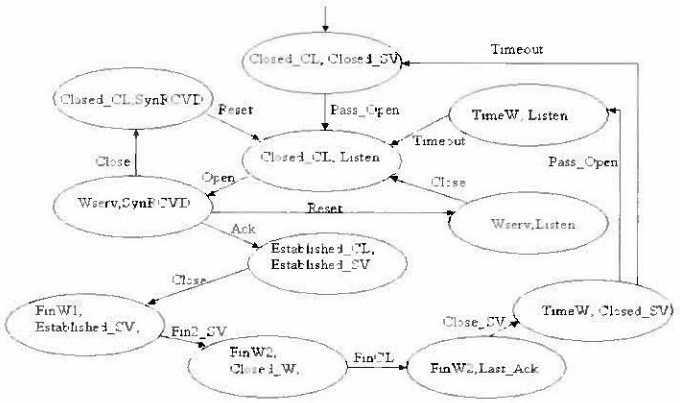

Figura 2.16 – MEF correspondente à Figura 2.15 gerada pela ferramenta PerformCharts [Amaral, 2005].

```
inicial( estado0).
trans( estado0, transicao0, estado1, L0, Ln ) :-
 receivel( 'PassOpen', L0, L1 ),
 transmit( L1, Ln ).
trans( estado3, transicao1, estado1, L0, Ln ) :-
 receivel( 'Timeout', L0, L1 ),
 transmit( L1, Ln ).
trans( estado4, transicao2, estado1, L0, Ln ) :-
 receivel( 'Close', L0, L1 ),
 transmit( L1, Ln ).
```

Figura 2.17 – MEF descrita como Base de Fatos – entrada para ferramenta CONDADO [Amaral, 2005].

```
Caso de teste #1:
senddata(L,PassOpen) recdata( )
senddata(L,Open) recdata(L,SynCL)
senddata(L,Reset) recdata( )
senddata(L,Close) recdata( )
senddata(L,Open) recdata(L,SynCL)
senddata(L,Close) recdata( )
senddata(L,Reset) recdata( )
senddata(L,Open) recdata(L,SynCL)
senddata(L,Ack) recdata(L,ServiceOK)
senddata(L,Close) recdata(L,FinCL)
senddata(L,Fin2SV) recdata(L,AckClose)
senddata(L,FinCL) recdata(L,FinSV)
senddata(L,CloseSV) recdata(L,Ack2CL)
senddata(L,PassOpen) recdata( )
senddata(L,Timeout) recdata( )
senddata(L,Open) recdata(L,SynCL)
senddata(L,Ack) recdata(L,ServiceOK)
senddata(L,Close) recdata(L,FinCL)
senddata(L,Fin2SV) recdata(L,AckClose)
senddata(L,FinCL) recdata(L,FinSV)
```

Figura 2.18 – Casos de Testes – saída da ferramenta CONDADO [Amaral, 2005].

2.7 Considerações Finais

As mesmas observações feitas na Parte A servem também para a geração de casos de teste. A modelagem em alto nível, como a obtida por Statecharts, vai ao encontro da representação de lógica complexa inerente a sistemas modernos atuais. A especificação por si só não resolve o problema. É necessário associar uma solução à especificação, como no caso tanto da avaliação de desempenho quanto da geração de seqüências de testes.

Nos trabalhos em andamento, os outros métodos (T, UIO e D) estão sendo incluídos na ferramenta. Também tem-se investigado o método *switch cover*, para que se possa implementá-lo como um método independente caso não se deseje utilizar a ferramenta CONDADO.

Como já foi mencionado na Parte A, a ferramenta PerformCharts será disponibilizada para utilização pela Internet. No caso específico de geração de testes, isso representaria uma grande vantagem, possibilitando organizar um trabalho colaborativo, já que a fabricação de satélites no INPE conta com a cooperação de outros países.

Referências

Amaral, A.S.M.S. **Geração de Casos de Testes para Sistemas Especificados em Statecharts**. Dissertação de Mestrado em Computação Aplicada. INPE, São José dos Campos, 2005

Beizer, B. **Software Testing Techniques**. International Thomas Publishing, Co., 1990

Binder, R.V. **Testing Object-Oriented Systems: Models, Patterns and Tools**. Addison-Wesley, USA, 1999

Burnstein, I. **Practical Software Testing: A Process-Oriented Approach**. Springer, USA, 2003

Chow, T.S. Testing Software Design Modeled by Finite State Machines. **IEEE Transactions on Software Engineering**, Vol. 4(3), pp. 178-187, 1978

IEEE **IEEE Standards for Software Test Documentation**. ANSI, 2004

Lee, D.; Yannakakis, M. Principles and Methods for Testing Finite State Machines – A Survey. **Proceedings of the IEEE**, Vol. 84(8), pp. 1090-1123, 1996

Myers, G.J. **The Art of Software Testing**. John Wiley & Sons, Inc., 2004

Pressman, R.S. **Software Engineering: A Practitioner's Approach**. McGraw-Hill, 2004

Santiago, V.; Amaral, A.S.M.S.; Vijaykumar, N.L.; Matiello-Francisco, M.F.; Martins, E.; Lopes, O.C. A Practical Approach for Automated Test Case Generation using Statecharts. *2nd International Workshop on Testing and Quality Assurance for Component-Based Systems (TQACBS)*. Chicaco, USA, 2006

Sidhu, D.P.; Leung, T. Formal Methods for Protocl Testing: A Detailed Study. **IEEE Transactions on Software Engineering**, Vol. 15(4), pp. 413-426, 1989