A ANÁLISE DE MUTANTES NO CONTEXTO DE SISTEMAS REATIVOS: Uma Contribuição para o Estabelecimento de Estratégias de Teste e Validação

Sandra C. Pinto Ferraz Fabbri

ou 0 k

Tese apresentada ao Instituto de Física de São Carlos, da Universidade de São Paulo, para a obtenção do título de Doutor em Ciências (Física Aplicada - opção Física Computacional)

Orientador: Prof. Dr. José Carlos Maldonado

São Carlos
1996

Fabbri, Sandra C. Pinto Ferraz
A Análise de Mutantes no Contexto de Sistemas Reativos:
Uma Contribuição para o Estabelecimento de Estratégias de
Teste e Validação/Sandra C. Pinto Ferraz Fabbri.
São Carlos, 1996.
237p.

Tese (Doutorado) - Instituto de Física de São Carlos, 1996.

Orientador: Prof. Dr. José Carlos Maldonado

1.Análise de Mutantes. 2.Sistemas Reativos.
3.Teste e Validação. I. Título.

UNIVERSIDADE DE SÃO PAULO
Instituto de Física de São Carlos

Av. Dr. Carlos Botelho, 1465
CEP 13560-250 - São Carlos - SP
Brasil

Fone (016) 272-6222
Fax (016) 272-2218

MEMBROS DA COMISSÃO JULGADORA DA TESE DE DOUTORADO DE **SANDRA CAMARGO PINTO FERRAZ FABBRI** APRESENTADA AO INSTITUTO DE FÍSICA DE SÃO CARLOS, UNIVERSIDADE DE SÃO PAULO, EM 21/10/1996.

COMISSÃO JULGADORA:

Prof. Dr. José Carlos Maldonado/ICMSC-USP

Profa. Dra. Beatriz Mascia Daltrini/UNICAMP

Prof. Dr. Paulo Roberto Freire Cunha/UFPe

Prof. Dr. Luciano da Fontoura Costa/IFSC-USP

Prof. Dr. Valentin Obac Roda/IFSC-USP

Ao Glauco e

ao Bruno

AGRADECIMENTOS

Ao meu orientador, Prof. Dr. José Carlos Maldonado, pela excelente oritentação, que mesmo de longe neste último ano, foi igualmente efetiva e incansável, não permitindo que a distância interferisse na conclusão deste trabalho. Agradeço também por sua amizade, apoio e incentivo constantes e por ter me cedido sua sala e recursos computacionais durante seu período de afastamento.

Ao Prof. Dr. Paulo Cesar Masiero, pela amizade, sugestões e discussões que tanto contribuíram para o desenvolvimento do trabalho.

À Profª. Dra. Rosely Sanches, pelo carinho, amizade e por sua constante disponibilidade.

À Profª. Dra. Sandra S. Godoy, pela troca de idéias.

À Rosângela, por sua amizade e apoio em todos os momentos e principalmente, pela grande ajuda que me deu em relação à disciplina que ministrei neste último semestre.

Ao Márcio, por estar sempre pronto a ajudar, pela amizade e pela revisão da redação do trabalho.

À Simone e à Elisa, pela amizade, apoio e auxílio sempre que necessário.

À Renata, pela amizade e empréstimo de sua sala quando esteve afastada.

Aos alunos de iniciação científica, William V. da Palma, Fábio Luis V. da Silva e Fábio Yoiti Tokura pela implementação da ferramenta Proteum-RS/FSM.

A todos os amigos conquistados durante esse período, Fabiano, Paulo, Tchelo, Marcelo, Plínio, Silvia, Rejane, Lairce e Marisa.

A todos os professores, funcionários e bibliotecárias do Instituto de Ciências Matemáticas de São Carlos, pela receptividade e amizade, tornando esse período bastante agradável.

À Wladerez e Ivani, pela atenção dedicada.

Ao Glauco, pelo apoio e incentivo e ao Bruno, um agradecimento especial por sua paciência e maturidade em aceitar as várias vezes que estive ausente de seu dia a dia.

A toda minha família, pelo constante apoio e à Irene, por todo carinho dispensado ao Bruno durante esses anos.

Ao Departamento de Computação da UFSCar.

Ao CNPq, CAPES e FAPESP pelo apoio financeiro.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Este trabalho propõe a extensão do critério Análise de Mutantes, originalmente desenvolvido para o teste de programas, para sua aplicação no teste de especificações do aspecto comportamental de Sistemas Reativos. Esses sistemas constituem hoje um componente fundamental em várias atividades humanas e, em geral, falhas nos mesmos podem envolver grandes riscos à vida ou ao patrimônio. Isso torna imprescindível um maior rigor no processo de desenvolvimento e, em particular, na atividade de teste, que é fundamentalmente baseada em simulação, não fornecendo critério que avalie essa atividade de forma quantitativa. A proposta aborda a aplicação da Análise de Mutantes na validação de especificações de Sistemas Reativos baseadas em três técnicas formais, que possuem apoio gráfico, mais utilizadas para este fim: Máquinas de Estados Finitos, Redes de Petri e Statecharts. Para a aplicação do critério nesse contexto, estabeleceu-se um paralelo entre os níveis de programa e de especificação, quanto às suas hipóteses básicas do programador competente e do efeito de acoplamento. Foram definidos os operadores de mutação para cada uma das três técnicas consideradas, além de critérios de mutação alternativa que visam à minimização no custo de aplicação do critério. Foram realizados, manualmente, dois experimentos com o objetivo de validar os mecanismos propostos. Um deles foi aplicado em Máquinas de Estados Finitos e o outro, em Redes de Petri. Os resultados mostram evidências do aspecto complementar do critério Análise de Mutantes em relação às formas disponíveis de teste de especificações existentes na literatura. Apresenta-se também um protótipo da ferramenta Proteum-RS/FSM que apóia a aplicação da Análise de Mutantes em Máquinas de Estados Finitos e discute-se a instanciação dessa ferramenta para apoiar a aplicação do critério nos contextos de Statecharts e Redes de Petri.

ABSTRACT

Reactive Systems are a fundamental component in several activities and failures in these systems may cause risks to life or financial losses. Thus, the development process and particularly the test activity must be carried on with extreme care. This work proposes the use of Mutation Analysis – a technique originally proposed at program level testing – in the context of testing and validation of specifications of Reactive Systems; three graphical techniques are considered: Finite State Machines, Petri Nets and Statecharts. Currently, the test of such specifications is mainly based on simulation. The relevance of a test adequacy assessment criterion for such activity has been recognized by many researchers and practitioners and constitutes the objective of this work. The application of Mutation Analysis in this context is based in the assumption that the basic hypothesis valid to the program level – the hypothesis of the competent programmer and coupling effect – also hold to the specification level. Mutation operators are defined for the three specification techniques. Also, alternative criteria were defined aiming at reducing the cost of application of Mutation Analysis in this context. Two experiments were manually conducted in order to validate the proposed ideas. The results show evidences of the complementary aspects among existent methods and Mutation Analysis for testing specifications of Reactive Systems. A prototype of Proteum-RS/FSM, a tool to support Mutation Analysis for Finite State Machines is presented and its extensions to Statecharts and to Petri Nets are discussed.

CAPÍTULO 1

INTRODUÇÃO

Neste capítulo situa-se, dentro do âmbito da Engenharia de Software, a pesquisa realizada neste trabalho. São apresentados o contexto em que o trabalho está inserido, sua motivação e seus objetivos. No final do capítulo encontra-se a organização do trabalho.

1.1 Contexto em que a Tese se Insere

Software é atualmente um componente básico e necessário em várias atividades humanas. Muitos dos sistemas desenvolvidos são bastante complexos e podem envolver riscos à vida humana, exigindo, portanto, um desenvolvimento criterioso e de alta qualidade, para que esses riscos sejam minimizados (Pressman, 1992).

Basicamente, pode-se classificar o software nas seguintes categorias: *software básico* (sistemas que dão suporte a outros programas; por exemplo, compiladores, editores, sistemas operacionais, etc.), *sistemas de informação* (sistemas que apóiam operações relacionadas a negócios ou tomadas de decisão; por exemplo, folha de pagamento, controle de estoque, etc.), *sistemas científicos* (caracterizados por algoritmos de manipulação numérica, simulação, CAD — "computer aided design", etc.), *sistemas embutidos* (relacionados ao mercado industrial; por exemplo, funções digitais num painel automobilístico), *sistemas pessoais* (caracterizados por funções que apóiam atividades pessoais; por exemplo, controle financeiro, banco de dados, processadores de texto, etc.);

sistemas de inteligência artificial (caracterizados pelo uso de algoritmos não numéricos, e maior dificuldade de automatização; por exemplo, reconhecimento de som e imagem, prova de teoremas, jogos, etc.) e *sistemas reativos* (cuja característica principal é interagir com o meio ambiente reagindo a estímulos) (Pressman, 1992).

Dentre essas classes de sistemas, os Sistemas Reativos possuem associados a eles um aspecto bastante crítico tendendo a atingir grande complexidade. Normalmente, esse tipo de sistema controla atividades humanas essenciais, de forma que falhas no mesmo podem envolver grandes riscos à vida ou ao patrimônio, como por exemplo, controle metroviário, controle de monitoramento hospitalar, controle de tráfego aéreo ou protocolos de comunicação. Especialmente nos casos onde o risco à vida humana é bastante alto, torna-se imprescindível um maior rigor no processo de desenvolvimento. Tal rigor é conseguido no uso disciplinado de técnicas especializadas, seguras e eficientes na produção de software, as quais estão definidas e agrupadas no que se chama *Engenharia de Software*, que evoluiu exatamente para atender às necessidades de desenvolvimento de produtos de qualidade a um baixo custo (Pressman, 1992).

A Engenharia de Software, embora possua várias definições, segundo Pressman (1992), é uma disciplina que pode ser vista de forma objetiva como o estabelecimento e uso dos princípios básicos de engenharia com a finalidade de se desenvolver software de uma forma econômica e que este seja confiável e eficiente. Ela agrega três elementos fundamentais: *métodos* (conjunto de atividades que fornecem uma base técnica para a construção do software), *ferramentas* (que dão suporte automatizado ou semi-automatizado aos métodos) e *procedimentos* (que estabelecem as ligações entre os métodos e as ferramentas). As formas como esses elementos estão agregados definem os diversos paradigmas da engenharia de software que, quando usados de maneira complementar, podem levar à obtenção de melhores resultados do que quando usados isoladamente. Independentemente do paradigma escolhido ou mesmo da área de aplicação, do tamanho do projeto ou da complexidade, o processo de desenvolvimento de software possui três fases genéricas: a *definição*, o *desenvolvimento* e a *manutenção* (Pressman, 1992).

A fase de *definição* tem como objetivo o *"o que"*, isto é, quais as informações a serem processadas, as funções e desempenho desejados, as interfaces que devem ser estabelecidas, as restrições do projeto e os critérios de validação que são requeridos. Nesta fase, três passos específicos são sempre realizados: a análise do sistema, o planejamento do projeto e a análise de requisitos.

A fase de *desenvolvimento* tem como objetivo o *"como"*, isto é, como devem ser projetadas a estrutura de dados e a arquitetura do software, como os procedimentos devem ser implementados, como o projeto deve ser traduzido para uma linguagem de programação e como o teste deve ser executado. Nesta fase, também são sempre realizados três passos específicos: o projeto, a codificação e o teste do software.

A fase de *manutenção* tem como objetivo as *mudanças*, sejam elas correções decorrentes de erros – manutenção corretiva, adaptações requeridas decorrentes de alterações no meio ambiente – manutenção adaptativa e melhoramentos relacionados a novas funções desejadas pelo usuário – manutenção perfectiva.

Acompanhando todo o processo de desenvolvimento, é realizado um conjunto de atividades de apoio, denominado *Garantia de Qualidade de Software*, para assegurar que a qualidade seja mantida em cada passo desse processo. A qualidade é um aspecto que deve ser tratado simultaneamente ao processo de desenvolvimento do software, pois ela não pode ser imposta depois que o produto está finalizado. Exemplos dessas atividades são: uso de métodos e ferramentas de análise, projeto, codificação e teste, realização de revisões técnicas formais, definição de uma estratégia de teste e controle de documentação e de mudanças realizadas no software.

Muitas dessas atividades são consideradas atividades de *verificação* e *validação*. O objetivo da verificação é assegurar que uma determinada função está sendo implementada corretamente enquanto que o objetivo da validação é assegurar que o software está sendo desenvolvido de acordo com os requisitos do usuário.

Das atividades de verificação e validação, a atividade de teste é considerada um elemento crítico para a garantia de qualidade do software. O desenvolvimento de técnicas e ferramentas que apóiem essa atividade é de fundamental importância e está diretamente relacionado à qualidade do teste. O principal objetivo do teste é encontrar erros, porém mesmo depois de terminada essa atividade, não se pode garantir a ausência de erros no software, uma vez que, em geral, o teste pode apenas evidenciar a presença de erros, mas não consegue provar sua ausência.

Em geral, o custo da atividade de teste é bastante alto em relação ao custo total de desenvolvimento. Assim, para se conseguir maior qualidade e produtividade na atividade de teste, têm sido definidos vários *critérios de teste* que atuam basicamente em nível de programas. Um critério estabelece requisitos que devem ser cumpridos na atividade de teste. Através da análise do quanto esses requisitos são satisfeitos – análise de cobertura – obtém-se uma maneira de quantificar essa atividade.

Os critérios de teste estão agrupados em três técnicas diferentes: a *funcional*, a *estrutural* e a *baseada em erros*. A principal diferença entre essas técnicas é a fonte de informação de onde são extraídos os requisitos de teste. Na funcional, os requisitos são extraídos da própria especificação do software; na estrutural, eles são extraídos de uma particular implementação e na baseada em erros, os requisitos são construídos com base nos erros típicos e comuns cometidos no processo de desenvolvimento. Um dos principais critérios da técnica de teste baseada em erros é o critério Análise de Mutantes, utilizado no contexto desta tese. Salienta-se que em nível de programas, os critérios de teste existentes são complementares e devem ser aplicados de maneira conjunta, para aumentar a qualidade da atividade de teste (Pressman, 1992).

Da mesma forma como se recomendam critérios para melhorar a qualidade do teste, a Engenharia de Software também recomenda que em todas as fases do processo de desenvolvimento, o projetista utilize métodos para apoiar seu trabalho visando aumentar a qualidade e minimizar a inserção de erros no software. Esses métodos são, em geral, classificados como métodos formais e informais. Os métodos informais não possuem rigor matemático e geram, normalmente, grande

ambigüidade. Os métodos formais são aqueles que possuem uma base matemática dada, em geral, por uma linguagem formal de especificação. Eles permitem que o projetista possa, de forma sistemática, especificar, desenvolver e verificar o software, ou parte dele. A base matemática fornece meios para definir precisamente noções de consistência, completitude e corretitude, além de possibilitar a verificação de propriedades do software sem a necessidade de executá-lo para mostrar seu comportamento. Em geral, os métodos formais são utilizados para especificar o aspecto comportamental e propriedades estruturais do software. Quando utilizados em fases iniciais do desenvolvimento do software, eles podem detectar falhas que só seriam reveladas nas fases de teste e depuração e, quando usados em fases mais adiantadas do desenvolvimento, ajudam a determinar a corretitude de uma implementação e a equivalência de diferentes implementações (Wing, 1990).

Como exemplos de técnicas utilizadas pelos métodos formais pode-se citar as três consideradas no contexto deste trabalho: Máquina de Estados Finitos (MEF) (Gill, 1962), Redes de Petri (Peterson, 1977) e Statecharts (Harel, 1987a, 1987b). Essas técnicas são intensamente utilizadas na especificação do aspecto comportamental de Sistemas Reativos que, como mencionado anteriormente, são sistemas que exigem, em geral, que as atividades de teste e validação sejam conduzidas com bastante rigor e critério.

Nesse sentido, para apoiar as atividades de teste de tais especificações, vários métodos e abordagens são propostos na literatura. Para especificações baseadas em Máquinas de Estados Finitos, existem vários métodos de geração de seqüências de teste (Fujiwara et al, 1991; Chow, 1978; Gonenc, 1970; Naito e Tsunoyama, 1981; Sabnani e Dahbura, 1988). No entanto, em geral, as MEFs não satisfazem os requisitos exigidos por esses métodos para sua aplicação. Para as Redes de Petri, existem algumas técnicas de análise sendo que a principal é a árvore de alcançabilidade; as outras técnicas envolvem equações matriciais. Essas técnicas são utilizadas, basicamente, para identificar as propriedades das redes (Peterson, 1981). Para Statecharts, são propostas várias abordagens de simulação, que são consideradas formas de validação de Sistemas Reativos em geral e que necessitam de ferramentas para que possam ser utilizadas (Harel, 1992). Uma questão que não é considerada por essas formas de

validação, mas que é relevante para a atividade de teste, é o aspecto de cobertura. Petrenko e Bochmann (1996) abordam esse aspecto em relação às especificações baseadas em Máquinas de Estados Finitos, salientando a relevância de pesquisas nessa direção, uma vez que com a análise de cobertura a qualidade da atividade de teste pode ser quantificada.

Embora existam lacunas a serem preenchidas, observa-se que vários métodos e técnicas têm sido propostos como resultado de uma preocupação constante e cada vez maior em resolver os problemas relacionados ao processo de desenvolvimento de software. Em particular, os métodos formais, como os citados anteriormente, que possibilitam o apoio de simuladores para o teste e validação das especificações neles baseadas, atendem às características consideradas por Harel (1992) como fundamentais para que a essência dos problemas relacionados ao desenvolvimento de software complexo seja atingida. Harel (1992) em seu artigo "Biting the Silver Bullet – Toward a Brighter Future for System Development" considera que "uma abordagem "vanilla" para modelar sistemas, juntamente com poderosas noções de executabilidade e geração de código, pode provocar um grande impacto na essência do desenvolvimento de sistemas complexos".

Assim, no sentido de contribuir com novos mecanismos para facilitar o desenvolvimento de sistemas complexos, este trabalho propõe critérios e ferramentas de apoio ao teste e validação de sistemas em nível de especificação, de forma que erros possam ser detectados em fases iniciais do desenvolvimento de software.

Este trabalho está inserido nas linhas de pesquisas que vêm sendo realizadas pelo Grupo de Engenharia de Software do ICMSC - Instituto de Ciências Matemáticas de São Carlos - USP. As principais linhas de pesquisa desse grupo dão ênfase às técnicas de especificação, teste e manutenção de software e à elaboração de ferramentas que apóiam essas técnicas.

1.2 Motivação

De acordo com o contexto em que esta tese se insere, relacionam-se alguns pontos que foram fundamentais para a condução desta pesquisa:

- Cada vez mais sistemas de software são utilizados em contextos e aplicações que envolvem atividades humanas;

- Assim como qualquer outra atividade humana, o desenvolvimento de software está sujeito a erros;

- Os Sistemas Reativos são cada vez mais essenciais à vida humana, mas ao mesmo tempo em que podem ajudar, podem também envolver grandes riscos, caso eles venham a falhar;

- Embora existam várias técnicas que apóiam a especificação de Sistemas Reativos, nota-se a necessidade de critérios e ferramentas para apoiar o teste de tais especificações;

- Medidas de cobertura que possibilitem quantificar a atividade de teste são extremamente relevantes para a avaliação dessa atividade;

- O desenvolvimento de ferramentas que apóiem as atividades de teste é de fundamental importância, uma vez que a aplicação dos critérios de teste existentes torna-se inviável se não for automatizada. Além disso, com o uso de ferramentas consegue-se diminuir a introdução de erros decorrentes de falhas humanas na atividade de teste, aumentando-se a qualidade e produtividade dessa atividade e, conseqüentemente, aumentando-se a confiabilidade no produto que está sendo desenvolvido.

1.3 Objetivos

O principal interesse desta tese é explorar a aplicação do critério de teste Análise de Mutantes, definido originalmente para o teste de programas, no teste de especificações do aspecto comportamental de Sistemas Reativos, com ênfase nas técnicas Statecharts, Máquinas de Estados Finitos e Redes de Petri.

Assim, além de estudar os princípios básicos da Análise de Mutantes, o interesse é estudar essas técnicas de especificação e identificar conjuntos de erros simples que possam modelar erros típicos relacionados a elas, possibilitando o estabelecimento de operadores de mutação para cada uma das técnicas.

Outro objetivo é explorar, também em nível de especificações, o uso de formas alternativas do critério Análise de Mutantes com a finalidade de reduzir o custo e o tempo de processamento associados a esse critério, de forma semelhante às pesquisas que vêm sendo conduzidas em nível de teste de programas (Wong et al, 1994b).

É também interesse deste trabalho permitir um suporte automatizado à aplicação da Análise de Mutantes no teste de especificações, com ênfase em Máquinas de Estados Finitos e Statecharts.

1.4 Organização do Trabalho

Neste capítulo foram apresentados o contexto onde este trabalho está inserido, sua motivação e seus objetivos.

No Capítulo 2 apresenta-se uma revisão bibliográfica que se concentra principalmente nas fases de especificação e de teste, abordando-se, especialmente, as três técnicas formais utilizadas para a especificação do aspecto comportamental de Sistemas Reativos: Máquinas de Estados Finitos, Redes de Petri e Statecharts.

No Capítulo 3 apresenta-se uma revisão detalhada do critério de teste Análise de Mutantes.

No Capítulo 4 apresentam-se a aplicação do critério Análise de Mutantes na validação de especificações baseadas em Máquinas de Estados Finitos, o conjunto dos operadores de mutação definidos para possibilitar essa aplicação e também o uso de critérios alternativos da Análise de Mutantes nesse contexto.

No Capítulo 5 apresentam-se a aplicação do critério Análise de Mutantes na validação de especificações baseadas em Redes de Petri, o conjunto dos operadores de mutação definidos para possibilitar essa aplicação e o uso de critérios alternativos da Análise de Mutantes também no contexto dessa técnica.

No Capítulo 6 apresenta-se a aplicação do critério Análise de Mutantes na validação de especificações baseadas em Statecharts, definindo-se para isso estratégias de teste que permitem abstrair componentes de um Statecharts, possibilitando que estes sejam testados separadamente. Definem-se também o conjunto de operadores de mutação para essa técnica e critérios alternativos de mutação que visam à redução do custo de aplicação do critério.

No Capítulo 7 apresenta-se a ferramenta de teste Proteum-RS/FSM, que apóia a validação de Máquinas de Estados Finitos através do critério Análise de Mutantes, e discutem-se aspectos de sua extensão para Redes de Petri e Statecharts.

No Capítulo 8 apresentam-se as conclusões do trabalho e sugestões para novas pesquisas.

No Apêndice A apresenta-se a gramática da Linguagem de Especificação de Statecharts (LES), definida por Fortes (1991), tendo sido utilizada neste trabalho para a especificação e implementação da ferramenta Proteum-RS/FSM.

CAPÍTULO 2

REVISÃO BIBLIOGRÁFICA

2.1 Considerações Iniciais

Como mencionado no capítulo anterior, existem várias classes de sistemas. Dentre elas, a que interessa dentro do contexto deste trabalho é a dos Sistemas Reativos. Um sistema reativo, segundo Harel (1987a) é caracterizado por ser baseado em eventos e por ter que reagir continuamente a estímulos internos e externos. Exemplos desses sistemas são: sistema de telefonia, de tráfego aéreo, de monitoramento hospitalar de paciente, de controle metroviário, dentre outros.

A especificação e o projeto de Sistemas Reativos complexos e de grande porte são considerados um problema constante no desenvolvimento dos mesmos. A grande dificuldade está em descrever o aspecto comportamental de uma forma clara e que seja ao mesmo tempo formal e rigorosa. Tal formalismo permite que as especificações possam ser simuladas e testadas.

As atividades de teste e validação dos Sistemas Reativos constituem um aspecto relevante que deve ser tratado com bastante rigor e critério. As formas de teste mais freqüentemente utilizadas para testar as especificações do aspecto comportamental dos Sistemas Reativos são alguns tipos de simulação que permitem ao usuário avaliar se a especificação está de acordo com o comportamento esperado do sistema (Harel, 1992).

A proposta feita neste trabalho concentra-se em técnicas/critérios para o teste do aspecto comportamental de Sistemas Reativos. O objetivo deste capítulo

é fazer uma revisão das técnicas de teste e das técnicas de especificação de sistemas mais pertinentes ao trabalho.

O capítulo está organizado da seguinte maneira: na Seção 2.2 apresentam-se as técnicas de teste e validação mais diretamente relacionadas a este trabalho. Na Seção 2.3 discutem-se as principais técnicas para especificação de sistemas, descrevendo-se com mais detalhes aquelas utilizadas neste trabalho, isto é, Máquinas de Estados Finitos, Redes de Petri e Statecharts. Na Seção 2.4 apresentam-se as considerações finais.

2.2 Técnicas de Teste e Validação

Esta seção comenta as principais técnicas e critérios de teste, com o objetivo de melhor situar o critério Análise de Mutantes, utilizado na proposta desta pesquisa.

Segundo Pressman (1992), a atividade de teste é um elemento crítico para a garantia da qualidade de software e representa a última revisão da especificação, projeto e código. Pressman ainda comenta que o teste de sistemas dos quais dependam vidas humanas pode custar de três a cinco vezes mais do que todos os outros passos do processo de desenvolvimento.

Embora a Engenharia de Software proponha diretrizes e apresente vários métodos para serem aplicados nos passos do processo de desenvolvimento de software, que têm por objetivo evitar que erros sejam introduzidos, ainda assim, erros permanecem, justificando a relevância da atividade de teste na tentativa de eliminá-los (Maldonado,1991).

Segundo Myers (1979), teste é o processo de execução de um programa com o objetivo de encontrar um erro. Pela definição, nota-se que a atividade de teste exige um componente executável para sua realização e por isso, essa atividade tem se concentrado, essencialmente, no teste de programas.

A atividade de teste é caracterizada em três níveis de aplicação: *teste de unidade*, que tem por objetivo o módulo, que é a menor unidade do projeto do software, procurando identificar erros de lógica e de implementação; *teste de integração*, que visa identificar erros na interação entre os módulos, e é uma técnica sistemática para integrar os módulos componentes da estrutura do software e *teste de sistema*, que é realizado após o teste de integração e visa identificar erros de funções e características de desempenho, que não estejam de acordo com a especificação.

Segundo Myers (1979), um teste bem sucedido é aquele que revela um erro ainda não descoberto. Portanto, se a atividade de teste for conduzida com sucesso, ela descobrirá erros no software. Por outro lado, caso não sejam descobertos erros, a atividade de teste não pode provar a ausência deles. Em outras palavras, na atividade de teste pode-se revelar erros, mas caso isso não aconteça, não se pode afirmar que eles não existam. Assim, com o objetivo de definir casos de teste que tenham alta probabilidade de revelar erros, duas questões tornam-se relevantes no contexto da atividade de teste:

Como os casos de teste devem ser selecionados ?
Como pode-se dizer que um programa foi suficientemente testado ?

Para que essas questões possam ser respondidas, são definidos os seguintes conceitos: 1) *método de seleção de dados de teste*, como um procedimento para escolher casos de teste e 2) *critério de adequação de dados de teste*, como um procedimento para avaliar um conjunto de casos de teste.

Existe uma forte correspondência entre esses dois conceitos e por isso, costuma-se utilizar um único termo – *"critério de teste"* – para designá-los (Maldonado, 1991). Essa correspondência existe, pois dado um critério de adequação *C*, existe um método de seleção *M* que estabelece: "selecione um conjunto de teste que satisfaça o critério *C*". De forma análoga, dado um método de seleção *M* existe um critério de adequação *C* que determina: "um conjunto de teste é M-adequado se ele exercita todos os elementos requeridos pelo método *M*".

Vários critérios de teste têm sido propostos com o objetivo de permitir que conjuntos de teste possam ser estabelecidos de forma rigorosa e sistemática. Tais conjuntos, embora representando apenas um subconjunto do domínio de entrada, ainda assim devem ser eficientes para detectar os erros existentes no software. Esses critérios são classificados em três técnicas de teste: *Funcional*, *Estrutural* e *Baseada em Erros*. Elas diferem na origem da informação para o estabelecimento dos requisitos de teste, como é apresentado em seguida.

- *A Técnica Funcional*:

Essa técnica trata o software como uma caixa cujo conteúdo é desconhecido e apenas os dados de entrada fornecidos e respostas geradas pelo programa é que podem ser visualizados. Por isso, ela é conhecida como *teste caixa preta*. É muito importante para essa técnica que a especificação do sistema esteja correta e de acordo com os requisitos do usuário (DeMillo, 1987). Nessa técnica, são verificadas apenas as funções do sistema, que são identificadas a partir da especificação, sem que sejam considerados detalhes da implementação. Alguns critérios dessa técnica são:

- *Particionamento de Equivalência*: divide o domínio de entrada de um programa em classes de equivalência, a partir das quais os casos de teste são derivados. Tem por objetivo minimizar o número de casos de teste, selecionando apenas um caso de teste de cada classe pois, em princípio, todos os elementos de uma classe devem se comportar equivalentemente. Na verdade, são definidas classes que se aproximam dessa característica pois, segundo Myers (1979), é impossível caracterizar exatamente as classes de equivalência.
- *Análise do Valor Limite*: complementa o critério anterior, exigindo casos de teste nos limites de cada classe de equivalência. Segundo Pressman (1992), os erros costumam ocorrer com maior freqüência nos limites dos domínios de entrada, o que torna esse critério relevante para o teste funcional.
- *Grafo de Causa-Efeito*: verifica o efeito combinado de dados de entrada. As causas (condições de entrada) e os efeitos (ações) são identificados e combinados em um grafo a partir do qual é montada uma tabela de decisão e, a partir desta, são derivados os casos de teste e as saídas.

- *Error Guessing*: nesse critério os possíveis erros são "adivinhados", baseando-se nas características do software e do domínio de entrada e os casos de teste são elaborados com base em uma lista dos erros relacionados.

Os principais problemas associados à técnica funcional são o fato dela estar sujeita às inconsistências que podem ocorrer na especificação e também a dificuldade de quantificar a atividade de teste, uma vez que não é possível garantir que partes essenciais ou críticas do programa sejam testadas.

- *A Técnica Estrutural*:

Nessa técnica, ao contrário da técnica funcional, os requisitos de teste estão fortemente baseados nos aspectos de implementação. Por isso, ela é conhecida como *teste caixa branca*. Diversos critérios dessa técnica utilizam uma representação de programa conhecida como *grafo de fluxo de controle* ou *grafo de programa*, que é um grafo orientado onde cada vértice representa um bloco indivisível de comandos e cada aresta representa um desvio de um bloco para outro (Myers, 1979). Através do grafo de programa são escolhidos os componentes que devem ser executados, caracterizando assim o teste estrutural. Os critérios estruturais baseiam-se em tipos de estruturas diferentes para determinar quais componentes têm sua execução requerida. Os principais critérios de teste estrutural são:

- *Critérios Baseados na Complexidade*: utiliza informação sobre a complexidade do programa para determinar os requisitos de teste. O critério mais conhecido dessa classe é o *Critério de McCabe*, que utiliza a complexidade ciclomática para derivar os requisitos de teste. Essencialmente, esse critério requer que um conjunto linearmente independente de caminhos do grafo de programa seja executado (Pressman, 1992).
- *Critérios Baseados em Fluxo de Controle*: usam apenas características do controle da execução do programa, como comandos ou desvios, para determinar quais estruturas são requeridas. Os mais conhecidos são os critérios *Todos-Nós*, *Todos-Arcos* e *Todos-Caminhos*. O critério Todos-Nós exige que a execução do programa passe pelo menos uma vez em cada vértice do grafo de fluxo ou, em outras palavras, que cada comando do programa seja executado

pelo menos uma vez. O critério Todos-Arcos exige que cada aresta do grafo do programa seja exercitado ao menos uma vez. O critério Todos-Caminhos é, em geral, impraticável e requer que todos os caminhos possíveis do programa sejam executados ao menos uma vez (Pressman, 1992).

- *Critérios Baseados em Fluxo de Dados*: associa ao grafo de fluxo informações sobre o *fluxo de dados* do programa, isto é, explora associações entre pontos do programa onde é atribuído um valor a uma variável (chamado de *definição* da variável) e pontos onde esse valor é utilizado (chamado de *referência* ou *uso* da variável). Com base nessas associações são determinados os caminhos a serem exercitados. Exemplo desses critérios são os critérios de Rapps & Weyuker (1985), entre eles o critério *Todos-Usos* que requer que pelo menos um caminho entre todas as associações definição-uso de cada variável seja exercitado.

Mais recentemente, várias iniciativas têm sido conduzidas para o desenvolvimento de ferramentas de apoio aos critérios baseados em fluxo de controle e fluxo de dados no meio acadêmico e no meio industrial em cooperação com universidades, além das ferramentas comerciais hoje existentes que apóiam essencialmente critérios baseados em complexidade e em fluxo de controle. A POKE-TOOL (Chaim, 1991), desenvolvida na Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas, apóia a aplicação dos Critérios Potenciais Usos (Maldonado, 1991), baseados na análise de fluxo de dados; a ASSET, desenvolvida na New York University e que suporta o teste de programas Pascal, apóia a aplicação dos critérios de Rapps & Weyuker (1985); a PROTESTE (Price e Zorzo, 1990) desenvolvida na Universidade Federal do Rio Grande do Sul, que apóia os mesmos critérios da ASSET e a ATAC, desenvolvida no Bell Communications Research (Horgan, 1990), em cooperação com a Purdue University.

- *A Técnica Baseada em Erros*:

Essa técnica utiliza informação sobre erros mais freqüentes no processo de desenvolvimento de software e sobre tipos específicos de erros que se desejam

localizar (DeMillo e Offutt, 1991). Os principais critérios que se enquadram nessa técnica são:

- *Semeadura de Erros*: alguns erros são inseridos artificialmente no programa e durante o teste, dentre os erros encontrados, verificam-se quais são erros naturais e quais são artificiais. Calculando-se a razão dos erros artificiais pelos naturais tem-se, teoricamente, uma indicação do número de erros naturais ainda existentes no programa. Alguns pontos podem ser questionados em relação a esse critério. Um deles é o tamanho do programa e, conseqüentemente, da razão calculada que podem ser questionáveis estatisticamente (Budd, 1981). O outro ponto é que para aplicar esse critério considera-se que os erros sejam distribuídos uniformemente no programa, o que na prática não é verdade. Pressman (1992) cita que uma região onde um erro é encontrado tem grande probabilidade de possuir outros erros. Um terceiro ponto é que alguns erros artificiais podem interagir com os naturais, fazendo com que os erros naturais sejam omitidos pelos erros semeados (Budd, 1981).
- *Análise de Mutantes*: tem por objetivo avaliar a qualidade de um conjunto de casos de teste *T* em relação a um programa *P*. Para isso, utiliza um conjunto de programas ligeiramente diferentes de *P*, denominados mutantes de *P*, e busca obter um conjunto de casos de teste que consiga revelar as diferenças de comportamento existentes entre *P* e seus mutantes (DeMillo, 1980). Como este trabalho dá ênfase ao critério Análise de Mutantes, ele é tratado com maiores detalhes no Capítulo 3.

Salienta-se que nenhuma dessas técnicas é completa, isto é, nenhuma delas, em geral, é suficiente para garantir a qualidade da atividade de teste. Segundo Pressman (1992), essas técnicas de teste se complementam e portanto, devem ser utilizadas em conjunto para assegurar uma melhor qualidade a essa atividade.

Salienta-se também que a utilização desses critérios é praticamente inviável sem o apoio de ferramentas e que vários trabalhos têm sido conduzidos com o objetivo de fornecer apoio automatizado aos mesmos. Como exemplo, pode-se citar a *Mothra* (DeMillo et al, 1988; Choi et al, 1989b), que é um ambiente de teste baseado na Análise de Mutantes para programas FORTRAN-77,

desenvolvido em conjunto pela Purdue University e Georgia Institute of Technology. As principais características dessa ferramenta são: interface com o usuário baseada em janelas; nenhuma restrição, a priori, quanto ao tamanho do programa a ser testado; projeto capaz de permitir ao sistema suportar outras linguagens além do FORTRAN e fácil acoplamento de ferramentas como gerador de casos de teste, verificador de equivalência e oráculo. A Mothra tem acoplada a si uma ferramenta chamada Godzilla (DeMillo e Offutt, 1991), que gera automaticamente casos de teste que se aproximam da adequação. Também baseada no teste de mutação, tem-se a CMS (Acree, 1980) desenvolvida no Georgia Institute of Technology, para programas Cobol.

Outra ferramenta que também apóia o uso do critério Análise de Mutantes é a *Proteum/C* (Delamaro, 1993), desenvolvida no Instituto de Ciências Matemáticas de São Carlos - USP, para o teste de programas desenvolvidos na linguagem C. Suas principais características são: interface gráfica; é uma ferramenta multilinguagem, isto é, ela possui algumas partes de seu código construídas de maneira genérica, podendo ser configurada para diversas linguagens de programação; permite o teste de subprogramas, isto é, pode-se testar apenas uma ou algumas funções que compõem a unidade e além disso, é um ambiente compilado, onde a execução do programa original e dos mutantes é feita através de código executável. Existe atualmente uma nova versão dessa ferramenta que permite o teste de programas no modo "script" (Delamaro e Maldonado, 1996b), onde o testador define os passos desejados na atividade de teste e submete à execução esse roteiro, dispensando dessa forma a interação com o testador. Outra evolução dessa ferramenta procura apoiar o teste de integração baseado na Análise de Mutantes (Delamaro et al, 1996).

2.3 Técnicas de Especificação de Sistemas

O objetivo desta seção é comentar, ainda que de forma resumida, os conceitos mais relevantes para este trabalho, relacionados às três técnicas formais, que possuem apoio gráfico, bastante utilizadas para a especificação do

aspecto comportamental de Sistemas Reativos e que foram alvos da proposta desta tese.

Existem várias técnicas para apoiar a especificação do aspecto comportamental dos sistemas. Davis (1988) faz uma comparação entre algumas delas salientando que durante a fase de especificação dos requisitos o comportamento esperado do sistema a ser construído deve ser especificado em detalhes e que isso normalmente é feito utilizando-se linguagem natural. No entanto, a linguagem natural é inerentemente ambígua, resultando em documentos também ambíguos, inconsistentes e incompletos, características estas indesejáveis em qualquer tipo de sistema, principalmente naqueles onde o aspecto crítico de riscos à vida humana é ainda maior.

O uso de métodos formais para especificar o aspecto comportamental, segundo Davis (1988), reduz o número de erros que podem permanecer latentes no software, pois as técnicas que os apóiam alertam o projetista para a maioria desses erros. Além disso, o documento gerado na especificação dos requisitos serve como base para todas as atividades de projeto e para o planejamento do teste do sistema.

Wing (1990) apresenta uma discussão sobre vários aspectos relacionados aos métodos formais. Segundo Wing, "os métodos formais usados no desenvolvimento de sistemas de computação são técnicas embasadas matemáticamente para descrever as propriedades dos sistemas". Portanto, para essa autora, os termos "métodos" e "técnicas" são utilizados como sinônimos. No contexto deste trabalho, considera-se método como um conjunto maior, composto de diretrizes e de uma ou mais técnicas a serem utilizadas no desenvolvimento do software. Como já foi mencionado, um método é formal se ele possui uma base matemática, dada por uma linguagem de especificação formal. O tipo da linguagem, isto é, se ela é gráfica ou baseada em lógica de primeira ordem, tem influência na forma de aplicação do método. (Wing, 1990) estabelece algumas características para classificação dos métodos formais, observando-se que estes podem combinar algumas delas

Uma característica separa os métodos formais em duas grandes classes: os *orientados a modelos*, que permitem definir o comportamento do sistema através da construção de um modelo em termos de estruturas matemáticas como tuplas, relações, funções, conjuntos e seqüências e os *orientados a propriedades*, onde o comportamento do sistema é definido indiretamente, através da declaração de um conjunto de propriedades, na forma de axiomas, que devem ser satisfeitas pelo sistema.

Os principais métodos classificados por Wing (1990) como métodos orientados a modelos são: Máquina de Estados Finitos (MEF) (Gill, 1962), Redes de Petri (Peterson, 1977), Statecharts (Harel, 1987a, 1987b), VDM - Vienna Development Method (Jones, 1986), Z (Spivey, 1988), CCS - Calculus of Communicating Systems (Milner, 1980) e CSP - Communicating Sequential Processes (Hoare, 1984). Os orientados a propriedades podem ser subdivididos em duas categorias: os axiomáticos (apoiados por linguagens como Iota, OBJ, Anna e Larch) e os algébricos (apoiados por linguagens como Clear e Act One). A linguagem de especificação LOTOS – Language of Temporal Ordering of Specifications (Bolognesi e Brinksma, 1987), é uma combinação da linguagem Act One e do método CCS, com alguma influência do método CSP.

Outra característica está relacionada ao aspecto visual. Métodos visuais são aqueles cuja linguagem inclui elementos gráficos. São citados: Redes de Petri, Statecharts, Linguagem Visual de Miró (utilizada para especificar restrições de segurança), e também alguns outros bastante utilizados para a especificação de sistemas, mas por não possuirem um formalismo matemático, são considerados métodos semiformais, como por exemplo, o método de Jackson (1983), HIPO - Hierarchy Input Processing Output (IBM, 1974), SD - Structured Design (Gane e Sarson, 1979), dentre outros (Wing, 1990). Esses métodos semiformais, conhecidos na prática como métodos estruturados, possuem a vantagem de facilitar a comunicação entre o projetista e o cliente pois, em geral, podem ser facilmente compreendidos por este. No entanto, por não possuirem um formalismo e combinarem vários tipos de notações (diagramas e textos), podem levar a ambigüidades. Na tentativa de combinar as vantagens dos métodos semiformais com as vantagens dos métodos formais, existem algumas propostas de integrar

métodos estruturados com técnicas de especificação formal, como por exemplo, o trabalho de Semmens et al (1992).

Alguns métodos possuem suporte para a execução das especificações. Nesse caso, a especificação pode ser usada pelo projetista como um feedback da própria especificação e permitir a elaboração de protótipos. Alguns métodos que satisfazem essa característica são Statecharts, OBJ, Prolog e Paisley (Wing, 1990).

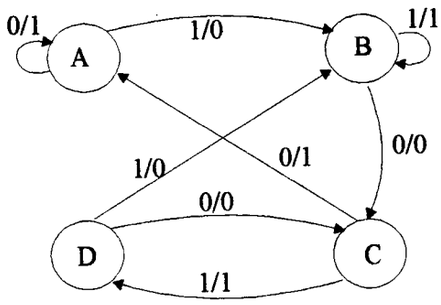
Furbach (1993) também comenta alguns métodos formais para a especificação de Sistemas Reativos. Além daqueles citados por Wing (1990), o autor cita também a linguagem ESTEREL (Boussinot e Simone, 1991), considerada uma linguagem bastante natural para a codificação de Sistemas Reativos.

No contexto deste trabalho dá-se ênfase às três técnicas de especificação formal, com apoio gráfico, que são comentadas mais detalhadamente em seguida.

2.3.1 Máquina de Estados Finitos (MEF)

É uma técnica de especificação que utiliza uma máquina, basicamente composta por estados e transições, que pode estar em somente um de seus estados num determinado momento. Em resposta a uma entrada, a máquina gera uma saída e muda de estado e, além disso, tanto a saída como o próximo estado são funções apenas do estado atual e da entrada. Ela é representada por um diagrama de transição de estado ou por uma tabela de transição (Figura 2.1). No caso do diagrama, círculos representam os estados e arcos direcionados representam as transições que podem ocorrer, levando de um estado a outro. O rótulo de cada arco representa a entrada e a saída gerada, no formato *e/s*. No caso da tabela de transição, as colunas representam os estados da máquina e as linhas, as possíveis entradas. Existem dois modelos que podem ser representados na tabela: o modelo de Mealy, onde a intersecção da linha com a coluna especifica o próximo estado e a saída gerada e o modelo de Moore onde a intersecção da

linha com a coluna contém apenas o próximo estado e existe uma coluna separada para indicar a saída associada a cada estado (Davis, 1988). O conjunto de *símbolos de entrada* é também denominado *alfabeto de entrada* ou *conjunto de eventos* e o conjunto de *símbolos de saída* é também denominado *alfabeto de saída*.



(a) Diagrama de Transição de Estado ou Máquina de Estado

Estado / Entrada	A	B	C	D
0	A/1	C/0	A/1	0/C
1	0/B	1/B	1/D	0/B

(b) Tabela de Tansição de Estado

Figura 2.1 - Exemplos de Representações de Máquinas de Estados Finitos

A aplicação da técnica Máquina de Estados Finitos não é vinculada a uma área científica específica, podendo ser utilizada em diversos contextos, servindo como uma aproximação de fenômenos físicos e abstratos (Gill, 1962). Essa técnica é muito utilizada na especificação do aspecto comportamental de Sistemas Reativos, particularmente, na área de protocolos de comunicação (Petrenko e Bochmann, 1996; Tan et al, 1996; Bochmann e Petrenko, 1994; Yao et al, 1994b).

Para a validação de especificações baseadas em Máquinas de Estados Finitos, são propostos, na literatura, vários métodos de geração de seqüências de teste, como os métodos W (Chow, 1978), Wp - "Partial W-Method" (Fujiwara et al, 1991), DS - "Distinguishing Sequences" (Gonenc, 1970), TT - "Transition-Tour" (Naito e Tsunoyama, 1981) e UIO - "Unique-Input-Output" (Sabnani e Dahbura, 1988). No entanto, para que eles possam ser aplicados, se faz necessário que a MEF satisfaça algumas restrições.

As propriedades e as principais características associadas às MEFs são apresentadas a seguir e podem ser encontradas com maior detalhe em Nakazato et al (1994a, 1994b, 1994c):

- *estados equivalentes:* o estado s_i da MEF M_1 e o estado s_j da MEF M_2, são equivalentes se quando exercitados por qualquer seqüência de entrada, produzem saídas idênticas; caso contrário, são denominados estados distinguíveis. M_1 e M_2 podem ser a mesma máquina.
- *especificação completa*: para cada estado existe uma transição para cada símbolo de entrada.
- *minimalidade*: nenhum par de estados é equivalente.
- *determinismo*: para cada entrada existe, no máximo, uma transição definida em cada estado.
- *conectividade forte*: para todo par de estados (s_i, s_j) existe uma seqüência de símbolos de entrada tal que essa seqüência leva a máquina do estado s_i para o estado s_j .

Além dessas propriedades existem também vários conceitos importantes nesse contexto, dentre eles, as seqüências básicas, pertinentes ao estabelecimento dos métodos de geração de seqüências de teste (Nakazato et al, 1994a, 1994b, 1994c):

- *seqüência de identificação*: é uma seqüência de símbolos de entrada que tem como objetivo identificar, de maneira única, cada um dos estados da máquina.

- *seqüência de sincronização*: é uma seqüência de símbolos de entrada que leva a máquina a um estado final especificado, independentemente da saída gerada ou do estado inicial.
- *seqüência distinguível*: é uma seqüência de identificação que produz seqüências de saída diferentes para cada um dos estados da máquina.
- *seqüência única de entrada/saída (UIO)*: é uma seqüência de identificação que relaciona um símbolo de entrada e um símbolo de saída e que é única para cada um dos estados da máquina.
- *conjunto de caracterização (W)*: é um conjunto de seqüências de identificação tal que os símbolos de saída observados após a aplicação dessas seqüências, numa ordem fixada, sobre cada um dos estados, são diferentes.

Com base nessas propriedades e seqüências, são estabelecidos os métodos de geração de seqüências de teste mencionados anteriormente, os quais são bastante utilizados no teste de conformidade de protocolos de comunicação. Em (Fujiwara et al, 1991) encontra-se uma avaliação desses métodos. Segundo os autores, o propósito de um método é o estabelecimento de um conjunto de casos de teste que apresentem as seguintes propriedades:

(1) geração de um conjunto pequeno, de modo que cada caso de teste seja executado rápido e facilmente;

(2) geração de um conjunto que cubra, o máximo possível, todos os possíveis erros contidos em uma implementação ou especificação.

Os métodos existentes diferenciam no tipo de compromisso alcançado entre esses dois objetivos conflitantes e na quantidade de formalismo que é utilizada na definição do método. Em seguida, apresenta-se uma breve definição dos métodos e dos requisitos exigidos para a aplicação dos mesmos:

- *Método TT*:

Esse método executa todas as transições ao menos uma vez, mas não se preocupa em identificar os estados destinos (Naito e Tsunoyama, 1981). Dado o objetivo do método, ele não requer que a MEF satisfaça qualquer restrição inicial.

- *Método DS*:

Esse método possui duas fases, sendo que na primeira é verificado se cada estado definido na especificação também existe na implementação e, na segunda fase, é verificado, para todas as transições definidas na especificação, se na implementação elas estão corretas em relação à saída e à transferência, isto é, ao estado destino (Gonenc, 1970). Ele teve como objetivo inicial a solução do problema de teste de um circuito de chaveamento seqüencial, a fim de determinar se o circuito operava de acordo com uma tabela de transição. Esse método exige que a MEF satisfaça os seguintes requisitos:

- conectividade forte;
- minimalidade;
- determinismo;
- seja uma máquina de Mealy;
- possua seqüência de sincronização e
- possua seqüência distinguível.

- *Método UIO*:

Esse método é baseado na idéia de que o teste deve visitar cada estado e cada transição do estado e deve verificar a existência de uma seqüência única (UIO) para cada estado, isto é, uma seqüência de entradas e saídas que possui como origem um único estado da máquina onde essa seqüência é válida (Sabnani e Dahbura, 1988). Esse método exige que a MEF satisfaça os seguintes requisitos:

- conectividade forte;
- minimalidade;
- determinismo;
- especificação completa;
- seja uma máquina de Mealy e
- possua estado inicial fixo.

- *Método W*:

Esse método, proposto por Chow (1978), tem como objetivo identificar todos os erros de seqüenciamento de uma máquina. Esses erros estão classificados em três categorias: erros de transferência, de operação e de estados extras/ausentes. O método está baseado na definição de dois conjuntos de seqüências de entrada, onde um deles representa um conjunto de cobertura das transições e o outro, representa um conjunto de caracterização. Assim, o conjunto W, das seqüências de teste pode distinguir entre os comportamentos de todo par de estados da especificação. Outros detalhes desse método são apresentados no Capítulo 4, onde essa classificação de erros proposta por Chow (1978) serviu de base para a definição dos operadores de mutação para Máquinas de Estados Finitos. Esse método exige que a MEF satisfaça os seguintes requisitos:

- minimalidade;
- determinismo;
- especificação completa;
- seja uma máquina de Mealy;
- possua estado inicial fixo;
- todo estado deve ser alcançável a partir do estado inicial e
- possua o conjunto de caracterização W.

- *Método Wp*:

Esse método é derivado do método W e sua principal vantagem sobre o método W é a redução no comprimento do conjunto de teste (Fujiwara et al, 1991). Em vez de ser utilizado o conjunto de caracterização do método W para checar cada estado s_i atingido, apenas um subconjunto dele pode ser utilizado em certos casos. Esse subconjunto, W_i , depende do estado atingido s_i , e é denominado conjunto de identificação de s_i. Esse método exige que a MEF satisfaça os mesmos requisitos do método W:

- minimalidade;
- determinismo;
- especificação completa;
- seja uma máquina de Mealy;
- possua estado inicial fixo;

- todo estado deve ser alcançável a partir do estado inicial e
- possua o conjunto de caracterização W.

Segundo a avaliação desses métodos apresentada em (Fujiwara et al, 1991), os conjuntos de teste derivados pelos métodos W, DS, UIO e TT detectam qualquer erro de saída na implementação desde que esta corresponda à especificação. No entanto, os erros de transferência, isto é, erros relacionados ao próximo estado atingido pela transição, nem sempre são detectados, a menos dos métodos W e DS, no caso do número de estados da implementação permanecer dentro de um certo limite.

Os autores ainda comentam que uma das maneiras de checar uma transição é verificar que ela atinge um próximo estado especificado. No caso dos métodos UIO e DS, o estado atingido é identificado pela saída obtida em resposta a uma única seqüência de entrada. No caso dos outros métodos, alguns estados requerem a aplicação de várias seqüências de entrada. Isso exige um retorno ao estado inicial após a aplicação das seqüências de entrada para o estado que está sendo testado.

Quanto à generalidade de aplicação, os métodos W e Wp são os únicos de aplicabilidade geral, no sentido de que o conjunto de caracterização W e os conjuntos de identificação W_i existem para todas as MEFs minimais. Por outro lado, as seqüências UIO e DS nem sempre existem, mesmo que as MEFs satisfaçam os requisitos de minimalidade, especificação completa e conectividade forte.

Os autores (Fujiwara et al, 1991) consideram que para a aplicação prática desses métodos de geração de seqüências de teste, duas questões são de fundamental importância:

(1) o comprimento do conjunto de teste gerado, isto é, o custo de executar o conjunto de teste, e

(2) a cobertura de erros, isto é, a confiança de que qualquer erro contido na implementação possa ser detectado com o conjunto de teste.

De maneira resumida, a resposta a essas questões pode ser observada pela Figura 2.2, ou seja:

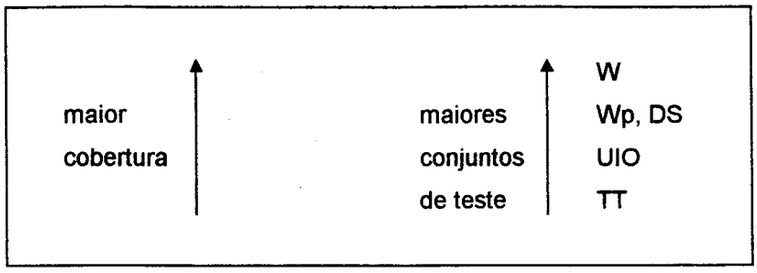


Figura 2.2 - Relação entre o Comprimento do Conjunto de Teste e a Cobertura dos Métodos de Geração de Seqüências de Teste para Máquinas de Estados Finitos (Fujiwara, 1991)

Fujiwara et al (1991) observam que esses métodos de geração de seqüências de teste para MEFs detectam todos os erros de saída e transferência na implementação, mas a aplicabilidade e o poder de detecção dos erros está vinculada aos requisitos exigidos, os quais nem sempre são satisfeitos. Observa-se, no entanto, que no caso de Máquinas de Estados Finitos Estendidas, como comentam Bochmann e Petrenko (1994), devem ser conduzidas pesquisas que apóiem o teste de especificações baseadas nessa técnica, pois como elas fazem uso de variáveis e ações, a atividade de teste é bastante relacionada ao teste de software. Os métodos citados não abordam todos os recursos utilizados pelas MEFs Estendidas.

Na Tabela 2.1 apresenta-se uma síntese dos requisitos exigidos pelos métodos de geração de seqüências de teste mencionados anteriormente, observando-se que o método TT não está relacionado pelo fato de não exigir nenhum requisito para sua aplicação.

Tabela 2.1 - Síntese dos Requisitos Exigidos pelos Métodos de Geração de Seqüências de Teste para Máquinas de Estados Finitos

Propriedade \ Critério	W	Wp	UIO	DS
Minimalidade	x	x	x	x
Especificação Completa	x	x	x	x
Conectividade Forte			x	x
Determinismo	x	x	x	x
Máquina de Mealy	x	x	x	x
Alcançabilidade do estado inicial	x	x		
Estado inicial fixo	x	x		
Seqüência de sincronização				x
Seqüência distinguível				x
Seqüência única de entrada/saída			x	
Conjunto de caracterização	x	x		

Em (Fujiwara et al, 1991) também é salientada a necessidade do desenvolvimento de ferramentas que apóiem a geração de casos de teste e analisem o conjunto de teste determinando a cobertura alcançada e permitindo que novos casos de teste sejam adicionados para explorarem aspectos particulares de comportamento.

Existem algumas ferramentas que apóiam o teste de protocolos de comunicação gerando casos de teste para especificações baseadas em Máquinas de Estados Finitos, como por exemplo, PT500 Protocol Tester, Ambiente OSTC – Open System Testing Consortium, CICA Portable Tester (os catálogos obtidos não fornecem informações sobre os métodos de geração de casos de teste utilizados). Além dessas, pode-se citar também a ferramenta TAG-Test Automatic Generation (Tan et al, 1996) que gera casos de teste automaticamente, implementando uma abordagem de identificação de transições similar ao conjunto de caracterização definido no Método W (Chow, 1978), usado para a identificação de estados.

2.3.2 Redes de Petri

Rede de Petri é uma técnica para o estudo de sistemas, sendo que sua aplicação ocorre através da modelagem. Muitas vezes um sistema não pode ser estudado diretamente, mas indiretamente, através de um modelo que o represente, normalmente em termos matemáticos, o que permite que esse modelo possa ser estudado e analisado. Os sistemas, em geral, são compostos de vários componentes que interagem, podendo cada um deles ser também um sistema. Cada componente tem seu próprio estado, o qual é uma abstração de uma informação que descreve sua ação, sendo que ele pode mudar ao longo do tempo. Os componentes de um sistema podem apresentar *concorrência* ou *paralelismo*, isto é, as atividades de um componente podem ocorrer simultaneamente com as atividades de outro. As Redes de Petri foram criadas para modelar esse tipo de sistema, com componentes que interagem concorrentemente (Peterson, 1977, 1981).

As Redes de Petri são compostas de quatro partes: um conjunto de *lugares* (places), um conjunto de *transições*, uma *função de entrada* (input) e uma *função de saída* (output). As funções de entrada e saída definem o mapeamento das transições para uma coleção de lugares denominada *lugares de entrada*, e das transições para uma coleção de lugares denominada *lugares de saída*, respectivamente. A *marcação* da Rede de Petri corresponde à atribuição de *tokens* (marcas) aos lugares da rede, o que permite a *execução* da mesma, através do *disparo* das transições. Uma transição pode ser disparada caso ela esteja *habilitada*, isto é, todos os seus lugares de entrada possuem tokens. Após o disparo de uma transição, os tokens são removidos dos lugares de entrada e são depositados em todos os seus lugares de saída.

Considerando a Rede de Petri da Figura 2.3, tem-se que: P1 a P7 compõem o conjunto dos lugares de entrada; t1 a t6 compõem o conjunto das transições; P3 e P7 compõem os lugares de entrada de t5; e P2 e P3 compõem os lugares de saída de t2.

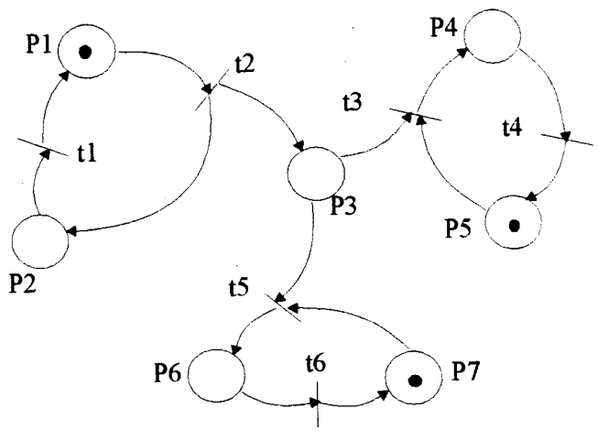


Figura 2.3 - Exemplo de uma Rede de Petri (Peterson, 1977)

Existem dois conceitos primitivos nessa técnica utilizados na descrição dos sistemas, que são os *eventos* e as *condições*. Os eventos correspondem às ações que acontecem no sistema e são controlados pelo estado do sistema. O estado do sistema, por sua vez, pode ser descrito como um conjunto de condições que podem ser verdadeiras ou falsas. Essa visão do sistema pode ser modelada pela Rede de Petri de forma que as condições correspondem aos lugares e os eventos, às transições.

As principais propriedades das Redes de Petri que auxiliam na modelagem de sistemas são: *concorrência* ou *paralelismo*, permitindo que dois eventos (transições) que não possuam interação e que estejam habilitados possam ocorrer de forma independente, não havendo necessidade de sincronizá-los. No entanto, quando a sincronização é necessária, é fácil modelá-la. Outra propriedade é a natureza *assíncrona* das redes, não havendo dependência do tempo para controlar a seqüência dos eventos, uma vez que a própria rede contém todas as informações necessárias para definir as possíveis seqüências. A execução da rede é vista como uma *seqüência de eventos* discretos, sendo que a ordem de ocorrência deles é uma dentre as muitas que possivelmente possam ser permitidas pela rede. Isso provoca um *não determinismo* na execução da rede mas, se num determinado momento, várias transições estão habilitadas, apenas

uma delas é disparada e a escolha é feita randomicamente. Esse aspecto introduz muita complexidade na análise do comportamento da rede. O não determinismo e o fato das transições não ocorrerem de forma simultânea podem levar a duas situações na modelagem de sistemas concorrentes: uma, de *concorrência*, em que é retratada uma situação em que os eventos podem ocorrer simultaneamente, isto é, a ocorrência de um não invalida a ocorrência do outro e apenas a ordem é que pode mudar, e a outra situação, de *conflito*, onde a ocorrência de um evento invalida a ocorrência do outro. Um aspecto também importante das Redes de Petri é a sua capacidade de modelar sistemas de forma *hierárquica*, onde uma rede pode ser substituída por um lugar ou por uma transição na modelagem de um nível mais abstrato do sistema, ou inversamente, um lugar ou uma transição pode ser substituída por uma rede, para apresentar um nível maior de detalhamento.

Inicialmente as Redes de Petri eram utilizadas apenas com um objetivo descritivo. No entanto, é necessário também analisar o modelo e para isso, devem ser definidas as propriedades a serem analisadas. As principais propriedades estão associadas com alguns tipos de redes:

- *safe net* (ou rede segura), quando cada lugar da rede pode conter apenas um único token;
- *k-bounded net* (ou rede k-limitada), quando cada lugar da rede pode conter até k tokens;
- *conservative net* (ou rede conservativa), quando o número de tokens na rede permanece constante.

Em geral, esses tipos de redes são utilizadas para modelar sistemas que fazem alocação de recursos. Um problema que pode surgir quando se faz alocação de recursos, é a situação de *deadlock*, isto é, uma situação onde o estado do sistema não é mais alterado pois a liberação de um recurso requer a alocação imediata de outro, o que não é possível, uma vez que todos os recursos disponíveis estão sendo utilizados. Um deadlock em uma Rede de Petri corresponde a uma transição ou um conjunto de transições que não podem disparar. Se uma transição não está relacionada com uma situação de deadlock, diz-se que a transição é *viva*. A maioria dos problemas que ocorrem no contexto das Redes de Petri, pode ser reduzida ao *problema de alcançabilidade* que

consiste em, dada uma Rede de Petri com uma marcação μ , verificar se uma determinada marcação μ' é alcançável a partir de μ .

As principais técnicas para analisar as Redes de Petri são a *árvore de alcançabilidade* e algumas outras técnicas baseadas em equações matriciais. Com a árvore de alcançabilidade, por exemplo, é possível decidir se a rede satisfaz as propriedades mencionadas anteriormente, isto é, se ela é segura, k-limitada ou conservativa. No entanto, em muitos casos, essa técnica não pode resolver o problema de alcançabilidade, nem o problema de determinar se uma transição é viva ou mesmo quais seqüências de transições são possíveis. O mesmo acontece com as equações matriciais, que nem sempre possuem uma solução e, às vezes, embora determinem como resultado uma seqüência de transições, esta não é válida a partir da marcação que ela foi gerada (Peterson, 1981).

Ainda que podendo ser aplicada para verificar algumas propriedades para determinados tipos de redes, como as redes limitadas, a árvore de alcançabilidade, na sua abordagem convencional, pode ser ineficiente ou intratável devido à explosão de estados em muitas aplicações práticas. Para contornar esse problema, Notomi e Murata (1994) propõem um método para construir um grafo de alcançabilidade hierárquico que permite a análise de alcançabilidade e de deadlock, para redes que, embora limitadas, a abordagem convencional não seria eficiente.

Um problema indecidível no contexto de Redes de Petri é a questão de equivalência entre duas redes, isto é, a verificação se os conjuntos de alcançabilidade de duas redes com marcação são subconjuntos um do outro. Outro aspecto em aberto é o problema da complexidade computacional associada à análise das Redes de Petri, a fim de determinar a viabilidade de modelar e analisar sistemas através dessa técnica.

Várias pesquisas têm sido conduzidas com o objetivo de aumentar o poder representacional das Redes de Petri, gerando outros tipos de redes, como por exemplo, as redes generalizadas, redes estendidas, redes coloridas, dentre outros. No entanto, como salientam Peterson (1981) e Murata (1989), à medida

que se aumenta o poder de representação, a capacidade de análise do modelo diminui.

2.3.3 Statecharts

Harel (1987a) apresenta a técnica Statecharts como sendo uma ampla extensão do formalismo das máquinas e dos diagramas de estado, baseada essencialmente em três elementos: hierarquia, concorrência e comunicação, que são aspectos essenciais para a especificação e projeto de sistemas complexos dirigidos a eventos.

Segundo Harel (1987b), é reconhecido na literatura o problema de especificar e projetar Sistemas Reativos grandes e complexos. Sistemas Reativos, em oposição aos sistemas transformacionais, são caracterizados por serem, na sua maioria, dirigidos a eventos e por reagirem continuamente a estímulos externos e internos. O grande problema está na dificuldade de especificar-se o aspecto comportamental de forma clara e, ao mesmo tempo, formal e rigorosa o suficiente para permitir sua simulação. O comportamento dos Sistemas Reativos é caracterizado pelo conjunto de seqüências possíveis de eventos de entrada e saída, condições, ações e, eventualmente, informações sobre restrições de tempo.

Statecharts é uma técnica de especificação do aspecto comportamental de Sistemas Reativos proposta por Harel (1987a, 1987b), que pode ser resumida da seguinte maneira:

Statecharts = diagramas de estado + decomposição +
ortogonalidade + broadcasting

Através dessa definição, pode-se observar que a técnica Statecharts é uma extensão de Máquinas de Estados Finitos, pois está fundamentada nos diagramas de estado, mas tem por objetivo solucionar os problemas inerentes das MEFs, que são: i) os diagramas de estado são planos, isto é, eles não possuem recurso para descrever hierarquia ou modularidade, não permitindo, dessa forma, um

desenvolvimento top-down ou bottom-up; ii) os diagramas de estado não facilitam a apresentação de transições quando essas são disparadas pelo mesmo evento mas saem de estados diferentes, isto é, todas elas devem ser denotadas explicitamente em MEF; iii) a utilização de diagramas de estados torna-se quase impossível à medida que o sistema cresce, pois o número de estados cresce exponencialmente; e iv) os diagramas de estado são seqüenciais e não representam, de forma natural, aspectos de concorrência (Harel, 1987b). Na Figura 2.4 apresenta-se um exemplo de um Statecharts.

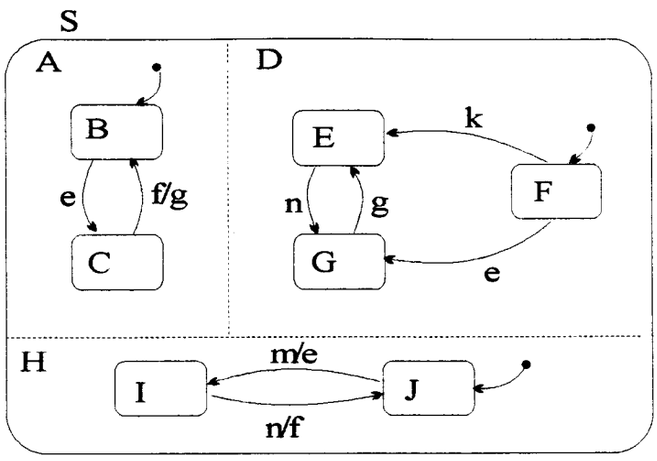


Figura 2.4 - Exemplo de um Statecharts (Harel, 1987b)

Em seguida, comentam-se algumas características da técnica Statecharts, que são ilustradas no Capítulo 6, a fim de facilitar a compreensão dos tópicos lá abordados.

A idéia de *decomposição* permite desenvolver o sistema de forma modularizada além de facilitar a representação das transições quando estas ocorrem partindo de diferentes estados, indo para um mesmo estado.

A *ortogonalidade* corresponde à especificação de estados do tipo AND, isto é, estados que são compostos de subestados paralelos, que ficam ativos simultaneamente. Ao contrário, em estados tipo OR, apenas um pode estar ativo em cada instante, como é o caso das MEFs. Quando existem estados do tipo AND, os aspectos de sincronização entre eles são representados por algumas

condições e eventos, que são avaliados no momento em que uma transição esteja habilitada, isto é, apta a ocorrer. Eles têm por objetivo avaliar ou confirmar a ocorrência de uma dada situação num componente paralelo que, se confirmada, possibilita que a transição seja disparada. Como exemplo, têm-se as condições *in(s)*, que é verdadeira caso o estado *s* esteja ativo, ou *not-yet(e)*, abreviado por *ny(e)*, que é verdadeira caso o evento *e* ainda não tenha ocorrido, e os eventos *exit(s)*, abreviado por *ex(s)*, que ocorre quando o estado *s* deixou de estar ativo, e *entered(s)*, abreviado por *en(s)*, que ocorre quando o estado *s* passou a estar ativo.

Segundo Harel (1987b), a forma de se modelar concorrência nos Statecharts é através de seus eventos de saída. Assim, os Statecharts podem ser vistos como uma extensão das máquinas de Mealy, uma vez que esses eventos, chamados de *ações*, podem ser atribuídos, opcionalmente, a um evento que provoca o disparo de uma transição. No entanto, em contraste às máquinas de Mealy, nos Statecharts as ações não são meramente enviadas, como simples saídas, para o exterior do sistema que está representado. Normalmente as ações afetam o comportamento do próprio Statecharts, nos componentes ortogonais, através de um mecanismo denominado *broadcasting* (reação em cadeia). A ação é considerada então como um *evento interno* que pode, possivelmente, causar outras transições em outros componentes. Na reação em cadeia, duas condições são relevantes: a condição *ny(e)*, mencionada anteriormente, e também a condição *current(c)*, abreviada por *cr(c)*, que refere-se ao valor corrente da condição *c* na reação em cadeia. Da mesma forma, tem-se *cr(v)* para representar o valor corrente da variável ou expressão *v* na reação em cadeia.

Observa-se que a *configuração* de estados do Statecharts, isto é, o conjunto de estados que estão ativos num determinado instante, não possui um comprimento fixo, mudando à medida que se entra ou sai dos estados ortogonais. Além disso, as reações em cadeia devem obedecer duas restrições, de certa forma, contraditórias: por um lado elas devem ter um tempo zero, mas por outro, a ordem das transições dentro das reações em cadeia é muito importante. Essa ordem estabelece o que se chama de *micro-passo*, que ocorre dentro de um *passo*, que é uma seqüência maximal do mesmo. As condições *cr* e *ny* são os

únicos mecanismos que detectam a ordem na qual os micro-passos ocorrem dentro de um passo.

Outro aspecto da técnica Statecharts é sua capacidade de "lembrar" alguns estados que foram visitados previamente, à qual se dá o nome de *história*. Quando um estado é ativado, o subestado dele que se torna ativo é, normalmente, o subestado default, a menos que esteja associado à transição que leva até esse estado, um símbolo de história *(H)*. Esse caso é denominado por Harel (1987a) de "entrada por história". Assim, em vez de ser ativado o subestado default, ficará ativado o último subestado que foi visitado anteriormente. Se um estado é decomposto em vários níveis e o aspecto de história for válido para todos esses níveis da hierarquia, essa história é denotada por *H**.

Dado o poder de representação dos Statecharts , identificam-se aplicações dessa técnica em vários contextos, como por exemplo, na modelagem de hipertextos (Zheng e Pong, 1992; Masiero et al, 1994b), em sistemas operacionais (Sowmya, 1992), em sistemas de aviação (Harel, 1992; Leveson et al, 1991) além de adaptação dessa técnica para sua aplicação dentro do paradigma de orientação a objetos (Coleman et al, 1992).

Somando-se ao poder de representação, os Statecharts também possuem sintaxe e semântica bem definidas, permitindo uma análise dos aspectos dinâmicos do modelo. De acordo com Harel (1992), verificar se o modelo está consistente e completo não previne a ocorrência de erros lógicos. É necessária a execução do modelo, cujo pré-requisito se resume na disponibilidade de uma semântica formal que contenha informação suficiente para definir de modo preciso o estado do sistema a cada passo da execução. Assim, os sistemas modelados através de técnicas que possuam esse formalismo, como é o caso dos Statecharts, são freqüentemente validados das seguintes formas: 1) *execução interativa*, que com o apoio de uma ferramenta automatizada, possibilita que o usuário represente o meio ambiente do sistema gerando, passo a passo, eventos que provocam alterações no estado do mesmo; 2) *execução em batch*, onde a execução é feita de forma iterativa e não mais interativa, com base em um conjunto de eventos pré-determinados; 3) *execução programada*, que permite analisar o modelo sob condições geradas randomicamente, e onde podem ser

definidos pontos de parada a fim de que a ferramenta execute determinadas ações quando ocorrem situações específicas; 4) *execução exaustiva*, que consiste, teoricamente, em executar o modelo gerando-se todos os possíveis eventos externos e todas as alterações nos valores das condições e variáveis; em geral, esse tipo de simulação é impraticável e deve ser considerada como uma alternativa no caso de partes pequenas, críticas e bem isoladas do modelo.

Uma das ferramentas de apoio à utilização da técnica Statecharts é o Statemate (Harel et al, 1990), uma ferramenta comercial, que possibilita a edição, simulação e análise de especificações baseadas nessa técnica.

Outra ferramenta de apoio à técnica Statecharts, similar ao Statemate, é o ambiente StatSim (Statecharts Simulator) (Masiero et al, 1991), desenvolvido no Instituto de Ciências Matemáticas de São Carlos, pelo grupo de Engenharia de Software. Ele permite a edição e simulação de Statecharts, tanto na forma textual como na forma gráfica.

No ambiente StatSim dispõe-se atualmente da simulação *interativa*, que permite que o usuário especifique, em cada passo, os eventos que devem ser disparados, criando cenários da execução, no modo gráfico, ou um "log" no modo textual; da simulação *batch*, onde é definida uma seqüência de eventos, que é executada pelo ambiente de forma animada, ou passo a passo, sob controle do usuário; da simulação *programada*, que é feita com base em um programa de controle, especificado previamente, com a ocorrência dos eventos definida através de distribuições probabilísticas, e da simulação *exaustiva*, baseada na árvore de alcançabilidade (Fortes, 1991; Batista, 1991; Boaventura, 1992; Cangussu, 1993). Como será visto no Capítulo 7, alguns módulos do ambiente StatSim foram reutilizados na implementação da ferramenta Proteum-RS/FSM, a fim de permitir a análise e simulação da MEF e de seus mutantes.

2.3.4 Teste e Validação de Especificações de Sistemas

Resumindo os aspectos de teste e validação dessas três técnicas utilizadas para a especificação do aspecto comportamental de Sistemas Reativos tem-se que: se por um lado, para as Máquinas de Estados Finitos existem na literatura alguns métodos de geração de seqüências de teste, por outro, esses métodos exigem que as MEFs satisfaçam uma série de requisitos para que eles possam ser aplicados, o que não acontece na prática, em grande maioria dos casos. Esse fato torna as seqüências geradas pelos métodos menos eficazes na detecção de erros. Assim, no contexto de Máquinas de Estados Finitos, a Análise de Mutantes pode agir como uma forma complementar de teste aos métodos disponíveis atualmente (Fabbri et al, 1993, 1994b). A proposta de aplicação da Análise de Mutantes no contexto de Máquinas de Estados Finitos é abordada no Capítulo 4, onde se apresentam também os resultados de um experimento realizado manualmente, com o objetivo de avaliar essa proposta. Algumas iniciativas de utilização da Análise de Mutantes no contexto de teste de protocolos, em modelos baseados em máquinas de estados, vêm sendo realizadas, como por exemplo, a proposta feita por Probert e Guo (1991), que aplica esse critério no teste de especificações em Estelle. Esse trabalho é comentado com mais detalhe no próximo capítulo.

Quanto à técnica Redes de Petri, os principais recursos que se têm disponíveis para a análise das especificações baseadas nessa técnica são a árvore de alcançabilidade e as equações matriciais, ambos com capacidade de determinar algumas propriedades das redes, mas ineficientes para avaliar vários aspectos, como por exemplo, se uma seqüência de transições é possível ou não. Além disso, ao contrário do acontece no contexto de Máquinas de Estados Finitos, não se encontrou na literatura métodos que gerem seqüências de transições para validar especificações baseadas em Redes de Petri. Assim, no contexto de Redes de Petri, a Análise de Mutantes pode auxiliar na elaboração de casos de teste de forma incremental, à medida que os operadores de mutação vão sendo selecionados de acordo com as características ambientais, criticalidade da especificação e recursos disponíveis (Fabbri et al, 1994c, 1995a). A proposta de aplicação da Análise de Mutantes na validação de Redes de Petri assim como os resultados de um experimento realizado manualmente para avaliar essa proposta são apresentados no Capítulo 5.

Quanto à técnica Statecharts, as formas mais freqüentemente utilizadas para validação das especificações baseadas nessa técnica, são os vários tipos de simulação, salientando-se que não existem critérios para seleção de casos de teste. Embora Statecharts seja uma extensão de Máquinas de Estados Finitos, os métodos de geração de seqüências de teste existentes para as MEFs, quando muito podem ser aplicados nos componentes do tipo OR do Statecharts pois estes correspondem a MEFs. Além disso, com os aspectos adicionais da técnica Statecharts, os modelos tornam-se mais complexos pois podem envolver paralelismo, concorrência e história. Uma técnica de análise dos Statecharts é a árvore de alcançabilidade proposta por Masiero et al (1994a) que apresenta extensões à árvore de alcançabilidade convencional utilizada para análise de Redes de Petri. No contexto da técnica Statecharts a Análise de Mutantes pode facilitar a atividade de teste, uma vez que esse critério é baseado em um modelo de erros, o qual pode ser definido de forma adequada para retratar tipos de erros que explorem todas as características da técnica. A proposta de aplicação da Análise de Mutantes no teste de Statecharts é apresentada no Capítulo 6.

Dada a flexibilidade do critério Análise de Mutantes, embora seja ele um critério de teste desenvolvido para o teste de programas, sua aplicação em diferentes contextos parece ser uma alternativa bastante viável, necessitando-se, no entanto, de ferramentas que apóiem essa aplicação.

Os benefícios decorrentes do uso de ferramentas que apóiam a atividade de teste, independentemente do nível em que essa atividade esteja sendo realizada, são indiscutíveis. Ciente desses benefícios, discute-se a factibilidade da implementação de ferramentas de teste que dêem suporte à proposta feita neste trabalho, exemplificando-se com um protótipo da ferramenta *Proteum-RS/FSM*, (Fabbri et al, 1994a, 1995b), que apóia o teste de Máquinas de Estados Finitos pelo critério Análise de Mutantes, apresentada com detalhes no Capítulo 7.

2.4 Considerações Finais

Este capítulo apresentou uma revisão dos principais conceitos relacionados às fases de teste e de especificação de sistemas, as duas fases do ciclo de desenvolvimento de software pertinentes a este trabalho.

Nota-se que em nível de programação, a atividade de teste encontra-se apoiada por vários critérios que têm sido elaborados com o objetivo de fornecer uma maneira sistemática e rigorosa para a elaboração de conjuntos de casos de teste que sejam eficientes para detectar os erros existentes no software. Lembra-se que mesmo depois dos testes, em geral, não se pode afirmar que um programa esteja correto. Assim, os critérios contribuem para aumentar a confiança de que o software desempenha as funções esperadas.

Mathur e Wong (1994) comentam que os critérios de teste visam avaliar a adequação dos conjuntos de casos de teste utilizados, contribuindo para decidir se o programa foi bem testado ou não, pois em decorrência das atividades de teste, espera-se obter uma maior confiança no produto que está sendo testado e que ele seja liberado com o menor número de erros possível.

Quanto ao teste em nível de especificações, nota-se que não se tem o mesmo apoio encontrado para o nível de programas. Particularmente para o teste de especificações de Sistemas Reativos, alguns tipos de simulação compõem a maneira mais utilizada para o teste das mesmas. Quando a especificação é baseada em Máquinas de Estados Finitos, por exemplo, dispõem-se de alguns métodos de geração de seqüências de teste que exigem a satisfação de alguns requisitos para sua aplicação, na maioria das vezes não satisfeitos.

Outros pontos comentados por alguns autores são a necessidade de quantificação do teste em nível de especificação, a possibilidade de serem explorados determinados tipos de erros e também a falta de ferramentas que apóiem a atividade de teste (Fujiwara et al, 1991; Petrenko e Bochmann, 1996).

Nesse sentido, a proposta de aplicação do critério Análise de Mutantes e da ferramenta Proteum-RS pode satisfazer, de certa forma, essas necessidades. Com a Análise de Mutantes, tem-se o escore de mutação que permite avaliar o teste de forma quantitativa e tem-se também a possibilidade de definirem-se novos operadores de mutação para que determinados tipos de erros possam ser explorados na atividade de teste. Com o desenvolvimento da ferramenta Proteum-RS, a atividade de teste pode ser apoiada de forma automatizada, aspectos que serão tratados detalhadamente nos próximos capítulos.

CAPÍTULO 3

ANÁLISE DE MUTANTES NA ESPECIFICAÇÃO DE SISTEMAS

3.1 Considerações Iniciais

No Capítulo 2 foram caracterizados os objetivos e atividades principais da Engenharia de Software, dando-se maior ênfase às atividades de especificação e teste de sistemas.

Do ponto de vista de especificação, técnicas com apoio gráfico, principalmente aquelas baseadas em notação formal, têm sido difundidas e desempenhado um papel relevante nas diversas etapas e atividades de Engenharia de Software (Presmann, 1992; Coleman et al, 1992; Furbach, 1993; Leveson et al, 1994; Stotts e Furuta, 1989). Como já foi mencionado, neste trabalho dá-se ênfase às técnicas que possuem apoio gráfico correntemente mais difundidas para a especificação de Sistemas Reativos: Máquinas de Estados Finitos (Gill, 1962), Redes de Petri (Peterson, 1981) e Statecharts (Harel, 1987a), assim como estratégias para validar tais especificações.

Do ponto de vista de teste, dá-se ênfase ao critério Análise de Mutantes, que é apresentado com maior detalhe neste capítulo. Ele tem sua origem no teste de unidade e diversos trabalhos têm mostrado ser este um critério promissor, dado o estado tecnológico atual. Diferentemente das outras técnicas e critérios de teste existentes, a Análise de Mutantes é um critério bastante maleável, pelo fato de

estar baseado na caracterização dos erros típicos e comuns cometidos por um programador/especificador, o que o torna um critério de grande aplicabilidade.

Mais recentemente, diversos pesquisadores têm procurado explorar o uso desse critério em outros níveis de teste. Por exemplo, Delamaro et al (1996) investigam o uso desse critério para o teste de integração. Outros autores têm conduzido investigações no contexto de protocolos de comunicação (Petrenko e Bochmann, 1996; Wang e Liu, 1993; Bochmann e Petrenko, 1994). O objetivo neste trabalho é fornecer mecanismos complementares aos atualmente utilizados para o teste e validação do aspecto comportamental de Sistemas Reativos.

O critério Análise de Mutantes foi desenvolvido inicialmente para o teste de programas e a principal restrição para sua utilização é o grande número de mutantes gerados e conseqüente custo computacional. Devido a este fato, vários estudos têm sido conduzidos com a finalidade de atacar esse problema, tornando a aplicação do critério mais eficiente. Uma das propostas nesse sentido é a redução do número de mutantes gerados, através de formas alternativas de aplicação do critério, como é apresentado por Wong et al (1994b).

No contexto deste trabalho, explora-se a aplicação da Análise de Mutantes em nível de especificações. Para tanto, as hipóteses básicas do critério – hipótese do programador competente e efeito de acoplamento – foram adequadas para esse contexto. Ciente do benefício alcançado pela aplicação dos critérios alternativos da Análise de Mutantes em nível de unidade, investiga-se, adicionalmente neste trabalho, o custo e "strength" da Mutação Alternativa – Mutação Aleatória e Mutação Restrita – em nível de especificações.

Este capítulo está organizado da seguinte maneira: na Seção 3.2 apresentam-se os conceitos do critério Análise de Mutantes, os principais trabalhos relacionados ao critério e também algumas aplicações do mesmo em outros contextos, que não o teste de unidade. Na Seção 3.3 apresentam-se o paralelo que foi estabelecido para que o critério Análise de Mutantes pudesse ser aplicado em nível de especificações de sistemas, os critérios de mutação alternativa propostos e a convenção adotada para a definição de operadores de

mutação no contexto deste trabalho. Na Seção 3.4, apresentam-se as considerações finais.

3.2 O Critério Análise de Mutantes

A Análise de Mutantes surgiu na década de 70 na Yale University e Georgia Institute of Technology (DeMillo et al, 1978) e tem se mostrado, através de trabalhos empíricos e teóricos, ser um critério atrativo para o teste de programas (DeMillo, 1980; Budd et al, 1980; Horgan e Mathur, 1992; Wong et al, 1994a; Wong et al, 1994b). Recentemente, com o avanço da tecnologia de hardware e da arquitetura de computadores, observa-se um maior esforço em torno da construção de ferramentas de apoio a esse critério (Choi e Mathur, 1993; Krauser et al, 1991; Mathur e Krauser, 1988; Delamaro, 1993), procurando sistematizar seu uso e sua aplicação.

Em seguida sintetizam-se os principais aspectos e conceitos do critério Análise de Mutantes, os quais encontram-se detalhados em (Delamaro e Maldonado, 1993; Delamaro, 1993).

Basicamente, a idéia do critério Análise de Mutantes (DeMillo et al, 1978) é criar a confiança de que um programa P está correto produzindo-se, através de pequenas alterações sintáticas, um conjunto de programas, $\Phi(P)$, chamados de ***mutantes de P***, e construindo-se casos de teste capazes de provocar diferenças de comportamento entre P e seus mutantes. Essas alterações são feitas com base em um conjunto de operadores denominados ***operadores de mutação***. A cada operador pode-se associar um tipo ou uma classe de erros que se pretende revelar no programa.

Um ponto importante da aplicação da Análise de Mutantes é a geração dos mutantes, ou seja, a escolha e definição dos operadores de mutação. Para determinar um conjunto de operadores que gere uma vizinhança finita de P, isto é, um conjunto $\Phi(P)$ finito, de forma que a atividade de teste seja praticável, o critério Análise de Mutantes apóia-se nas hipóteses: ***do programador competente***, que afirma que um programa produzido por um programador competente ou está

correto ou está próximo do correto e do ***efeito de acoplamento*** que considera que dados de teste capazes de distinguir os mutantes que diferem de um programa correto somente em erros simples, são tão sensíveis que também distinguem, implicitamente, os mutantes com erros mais complexos. A noção de se restringir o conjunto de programas mutantes pode ser vista como uma delimitação dos tipos de erros que se deseja considerar. Em decorrência dessa maleabilidade na definição e escolha dos operadores de mutação, o critério Análise de Mutantes torna-se um critério atrativo, de fácil ajuste, para atender aos requisitos do sistema quanto ao custo e criticalidade.

A Análise de Mutantes consiste de 4 etapas principais:

(1) *Execução de P com base em um conjunto de casos de teste T*: esse passo consiste em executar o programa P utilizando um conjunto de teste T selecionado e verificar se para esse conjunto de teste o programa P produz os resultados esperados. Se um erro for revelado, P deve ser corrigido e a atividade de teste deve ser retomada posteriormente.

(2) *Geração de mutantes*: constrói-se a vizinhança $\Phi(P)$, dos programas mutantes, com base no conjunto de operadores de mutação. Como essa vizinhança deve ser finita para que o teste seja praticável, torna-se fundamental a escolha e determinação do conjunto de operadores de mutação. Esse é, portanto, o ponto chave do critério Análise de Mutantes e o sucesso de sua aplicação depende totalmente de $\Phi(P)$, que deve possuir as seguintes características: ser abrangente o suficiente para que se consiga revelar o maior número de erros possível em P e ter uma cardinalidade pequena para que seja tratável o problema de gerar e de verificar o comportamento dos mutantes.

É importante observar que existe a possibilidade de gerar-se mutantes a partir de k alterações simultâneas no programa P que está sendo testado. Eles são chamados de mutantes de ordem k. Alguns trabalhos, como (Budd et al, 1980; Offutt, 1992), mostram que mutantes de ordem superior a um ($k>1$), além de não contribuírem de forma significativa para a construção de casos de teste melhores, têm um custo de geração e execução demasiado alto. Portanto, em nível de

programa, tem-se utilizado na Análise de Mutantes uma vizinhança composta apenas dos mutantes de primeira ordem.

(3) *Execução dos mutantes*: Todos os mutantes são executados usando-se os casos de teste T como entradas. Se um mutante P_i apresenta resultados diferentes de P diz-se que esse mutante está *morto*; nesse caso, T conseguiu identificar o "erro" modelado pelo mutante ou, mais precisamente, conseguiu revelar a diferença entre P e P_i. Por outro lado, se P_i apresenta respostas idênticas a P, diz-se que ele continua *vivo*. Isto pode ocorrer por dois motivos: porque T não contém casos de teste capazes de distinguir P_i de P ou porque ambos os programas executam as mesmas funções, ou seja, são equivalentes. No primeiro caso, novos casos de teste podem ser adicionados a T para matar o mutante. No caso de mutantes equivalentes, nenhum caso de teste será capaz de distinguí-los, pois seus resultados são sempre iguais aos de P. O objetivo é achar um conjunto de casos de teste que consigam matar todos os mutantes não equivalentes.

Um ponto importante destacado em (DeMillo, 1980) é que a Análise de Mutantes fornece uma medida objetiva do nível de confiança na adequação dos casos de testes analisados. Após a execução dos mutantes, é possível avaliar a adequação dos casos de teste através do ***escore de mutação*** ("mutation score"), que relaciona o número de mutantes gerados com o número de mutantes mortos, e que, como conseqüência, fornece uma medida da confiabilidade do programa testado.

Dado o programa P e o conjunto de casos de teste T, define-se o escore de mutação *ms(P,T)* da seguinte maneira:

$$ms(P,T) = \frac{DM(P,T)}{M(P) - EM(P)} \qquad (1)$$

onde:

DM(P,T): número de mutantes mortos pelos casos de teste em T

M(P): número total de mutantes gerados

EM(P): número de mutantes gerados equivalentes a P

Note-se que apenas DM(P,T) depende do conjunto de casos de teste utilizado. Apesar disto, não se conhece, antecipadamente, o número de mutantes equivalentes gerados. EM(P) é obtido iterativamente à medida que o testador decide ou decide-se automaticamente, através da aplicação de heurísticas, marcar como equivalente um mutante M.

(4) *Análise dos mutantes*: nesse passo, o que se faz em primeiro lugar é decidir sobre a continuidade ou não da atividade de teste. É o passo que mais requer a intervenção humana, pois o testador deve avaliar o escore de mutação e julgar se a atividade de teste pode ser encerrada e, conseqüentemente, se o conjunto de casos de teste T é um bom conjunto de teste para o programa P. Caso o testador julgue ser necessário dar continuidade à atividade de teste, os mutantes que permaneceram vivos devem ser analisados com a finalidade de determinar se eles são ou não equivalentes a P. Em caso positivo, eles são descartados pois nunca haverá um caso de teste capaz de diferenciar os comportamentos de P e do mutante. No entanto, se o mutante não for equivalente, significa que é possível criar um caso de teste capaz de matar o mutante. Assim, o conjunto de teste T é melhorado, e os passos 1, 3 e 4 devem ser executados novamente até que se consiga um bom conjunto de casos de teste, ou seja, que o escore de mutação esteja o mais próximo de "1", mostrando que P não contém a maioria dos erros retratados nos mutantes gerados.

Ressalta-se que o critério Análise de Mutantes pode ser utilizado para dois propósitos distintos: como um critério de adequação, isto é, para verificar se um conjunto de teste T é adequado à Análise de Mutantes, ou como um critério de geração de casos de teste, para construir um conjunto de teste T que seja capaz de identificar os erros caracterizados pelos mutantes.

3.2.1 Trabalhos Relacionados ao Critério Análise de Mutantes

Nesta seção comentam-se alguns trabalhos relacionados com a aplicação do critério Análise de Mutantes em nível de unidade. Como esse critério tem se

mostrado bastante efetivo no teste de programas, várias pesquisas têm sido conduzidas no sentido de tornar viável a aplicação do mesmo, uma vez que, devido ao grande número de mutantes gerados, seu custo e sua complexidade computacional são bastante elevados. Assim, esses trabalhos procuram solucionar ou propor alguma alternativa para esses problemas.

O critério Análise de Mutantes é fortemente relacionado a um método clássico para detecção de erros lógicos em circuitos digitais, denominado *modelo de teste de falha única* ("single fault testing model") (Friedman, 1975). Esse modelo considera que várias falhas como curto-circuitos ou circuitos abertos manifestam-se em nível lógico na forma de linhas presas em 0 ou em 1. Ele assume que apenas uma linha de cada vez pode estar defeituosa, isto é, presa em 0 ou em 1 e então constrói combinações dos sinais de entrada capazes de revelar essas condições de erro. Esse método assume ainda a hipótese de que testes construídos dessa forma são, na prática, bons também para detectar erros múltiplos. Com base nessa consideração, em geral, o número de testes necessários para validar um circuito é bem inferior ao número de testes requerido para que o circuito seja testado exaustivamente, o qual cresce exponencialmente em relação ao número de linhas de entrada.

Fazendo uma analogia com o critério Análise de Mutantes, dois pontos podem ser ressaltados: 1) cada circuito que contém uma linha presa corresponde a um mutante que deve ser morto por uma entrada que provoque comportamentos diferentes entre o mutante e o circuito original; 2) a hipótese de que os testes são eficientes para detectar erros únicos e erros múltiplos corresponde ao efeito de acoplamento no critério Análise de Mutantes.

Vários trabalhos têm sido conduzidos com o objetivo de superar um dos maiores problemas associados à utilização do critério Análise de Mutantes: sua complexidade computacional, devido ao grande número de mutantes gerados, o que leva a um grande número de execuções desses mutantes para que o teste do programa seja realizado.

Procurando explorar os avanços ocorridos no desenvolvimento de novas arquiteturas de hardware, com máquinas de alto desempenho, vários trabalhos

têm sido propostos na tentativa de tornar economicamente viável a utilização do critério Análise de Mutantes. Esses trabalhos estão na linha de diminuição do tempo de execução dos mutantes, como por exemplo, a abordagem vetorizada de processamento, proposta por Mathur e Krauser (1988), a execução dos mutantes de maneira concorrente, utilizando o modelo de máquinas SIMD (Single Instruction, Multiple Data Stream) e a alternativa proposta por Choi et al (1989a), onde é descrita uma implementação de um sistema de mutação que utiliza uma arquitetura de hipercubo para distribuir a execução dos mutantes.

Além desses trabalhos, existem também soluções através de software. Uma dessas propostas, dada por Offutt e King (1987), considera que cada mutante difere do programa original em um único ponto e que até esse ponto a execução de ambos é igual, não sendo necessário que a execução desse trecho comum seja realizada duas vezes. Assim, a execução de cada mutante é iniciada somente a partir do ponto em que o mutante difere do programa original, utilizando a execução interpretada.

Uma outra proposta, feita por Weiss e Fleyshgakker (1993), define uma estrutura de dados, denominada MDS ("mutant data structure"), que armazena toda informação importante sobre a execução de um programa com um caso de teste. O objetivo da estrutura MDS é facilitar a decisão relativa à adequação de um conjunto de teste em relação à mutação forte (onde o comportamento do mutante é observado após o final da execução do programa). Pelo fato de um mutante ser morto pela mutação forte apenas se ele for morto pela mutação fraca (onde o comportamento do mutante é observado logo após o ponto onde ocorreu a mutação), o que pode ser decidido quando o local alterado é executado pelo caso de teste, então decidir sobre a adequação em relação à mutação forte, se reduz, primeiramente, a decidir sobre a adequação à mutação fraca, o que é mais simples, com a utilização da estrutura proposta. Os mesmos autores propõem um outro algoritmo em (Fleyshgakker e Weiss, 1994), denominado LMA ("lazy mutant analysis") que também tem por objetivo determinar se um mutante é morto pela mutação forte, por um determinado caso de teste, verificando se ele é morto por esse caso de teste pela mutação fraca. A diferença desse algoritmo é que ele evita a execução de muitos mutantes, determinando dinamicamente classes de

mutantes que possuem o mesmo comportamento e executando apenas representantes dessas classes.

Também com o intuito de viabilizar a aplicação do critério Análise de Mutantes, existem alguns trabalhos na linha de diminuição do número de mutantes gerados. Essa abordagem é completamente diferente das anteriores e tenta selecionar apenas um subconjunto dos mutantes para ser gerado, a fim de diminuir o número de execuções necessárias para a realização do teste, procurando manter a qualidade do teste.

Uma das estratégias nessa linha é apresentada por Marshall et al (1990) que propõem que sejam descartados os mutantes que apresentam anomalias de fluxo de dados. Uma anomalia de fluxo de dados corresponde à ocorrência de uma seqüência suspeita de ações sobre uma variável, isto é, uma seqüência que pode indicar um erro no uso de uma variável. Assim, antes da execução dos mutantes pode ser feito um pré processamento para localizar estaticamente essas anomalias e eliminar os mutantes que as possuam. A principal crítica a esse método é que muitos programas corretos possuem anomalias de fluxo de dados intencionais. Nesse caso, a proposta dos autores é que fique por conta do testador a verificação de possíveis anomalias intencionais para que este permita que os mutantes gerados também as possuam.

Offutt e Rothermel (1993) apresentam os resultados de uma série de experimentos utilizando Mutação Seletiva ("Selective Mutation"). A diferença desse critério comparando-o com a Análise de Mutantes, é que nele não são aplicados os operadores que normalmente são responsáveis pelo maior número de mutantes. O critério é referenciado como *N-seletivo*, no caso em que deixa de gerar os mutantes para *N* operadores. Essa abordagem foi aplicada em programas desenvolvidos em FORTRAN e os resultados obtidos mostram que essa estratégia é bastante atrativa como alternativa para a Análise de Mutantes, necessitando apenas que os resultados obtidos sejam confirmados no âmbito de outras linguagens de programação.

Wong et al (1994b) comentam que embora o teste de mutação tenha se mostrado efetivo na detecção de erros, o número de mutantes gerados pode ser

proibitivo mesmo para programas pequenos. Todos os mutantes precisam ser compilados, executados e analisados pelo testador, para que este decida sobre a equivalência para os mutantes que permanecem vivos. Assim, são avaliados, através de um experimento que aplica os critérios em um conjunto de programas selecionados, dois tipos de mutação alternativa que reduzem o número de mutantes gerados:

(1) *mutação aleatória*: que examina uma pequena porcentagem de mutantes, selecionados aleatoriamente de cada tipo de mutação (Acree, 1980).

(2) *mutação restrita*: que seleciona apenas alguns tipos de operadores de mutação.

A comparação desses critérios alternativos em relação à mutação completa – "full mutation" – (ou mutação forte, ou Análise de Mutantes) e em relação ao critério todos-usos (Rapps e Weyuker, 1982; Rapps e Weyuker, 1985), um dos critérios da técnica estrutural baseada em fluxo de dados, é feita com base no experimento, em termos de três fatores: custo, dificuldade de satisfação e eficácia na detecção de erros. O custo está relacionado ao trabalho necessário para satisfazer o critério; a dificuldade de satisfação ou "strength" relaciona-se à probabilidade de satisfazer um critério dado que foi satisfeito um outro e a eficácia na detecção de erros refere-se à capacidade de detecção de erros do critério.

Os autores (Wong et al, 1994b) avaliam o custo relacionado à mutação alternativa através de duas métricas: uma refere-se ao número de casos de teste requeridos para satisfazer o critério e a outra, ao número de mutantes a serem examinados. A dificuldade de satisfação é medida em termos da cobertura alcançada por um conjunto de teste para um dos critérios, dado que o conjunto satisfaz o outro. Já a eficácia na detecção de erros é determinada por uma métrica que computa a porcentagem de conjuntos de casos de teste adequados a cada um dos critérios que são capazes de revelar erros.

Como resultado do experimento, os autores mostram que quando comparados com a mutação completa, os critérios de mutação alternativa obtiveram uma redução significativa no custo em termos do número de casos de teste requeridos e também no número de mutantes a serem examinados, com uma pequena redução no fator "strength", referente à habilidade em distinguir

mutantes não equivalentes. Quando comparados com o critério todos-usos, eles requerem mais casos de teste e sofrem uma pequena perda no fator "strength" na habilidade em cobrir os requisitos do critério todos-usos. Além disso, a mutação alternativa demonstrou uma maior eficácia e um maior "strength" na habilidade de distinguir mutantes não equivalentes do que o critério todos-usos. Experimento semelhante foi conduzido por Souza (1996) onde, além de ser feita uma avaliação do critério Análise de Mutantes, é apresentada também uma comparação desse critério com os critérios Potenciais-Usos (Maldonado, 1991), em relação aos fatores de "strength" e custo.

Outros critérios, derivados da Análise de Mutantes, também foram criados com o intuito de solucionar os problemas apresentados por ela. Duncan e Robson (1990) propõem um mecanismo lógico, denominado *mutação ordenada* ("ordered mutation"), para direcionar o teste, a fim de alcançar completamente o teste de mutação, reduzindo a sobrecarga de processamento inerente ao mesmo. Consideram que é viável a construção de uma ferramenta mais eficiente, direcionando a geração de mutantes por blocos juntamente com uma hierarquia de operadores de mutação e um esquema de cobertura dos dados de teste. Esses aspectos podem ser resumidos da seguinte maneira:

(1) *ordenação dos dados de teste* - o critério Análise de Mutantes utiliza os dados de teste com uma determinada ordenação, depois que são identificadas as entradas que matam um determinado mutante, sempre que o programa em teste sofrer uma alteração e o processo de teste precisar ser repetido;

(2) *ordenação dos operadores de mutação* - os operadores podem ser ordenados a fim de permitir uma elaboração mais rápida e eficiente dos dados de teste; por exemplo, a troca de um operador relacional por outro pode seguir uma hierarquia caso o operador inicial seja "=", pois trocando-o por "≠" o caso de teste eliminaria também as trocas por "<", ">", etc.;

(3) *ordenação das aplicações mutantes* - considera o impacto das mutações construindo uma árvore de impacto dos blocos do programa em teste, definindo, para cada bloco, seus antecessores e sucessores e submetendo a um determinado mutante apenas o caso de teste que afeta (atravessa) esses blocos.

O método em si consiste em instrumentar o programa em teste para obter informação sobre a "travessia" dos blocos, armazenando informações sobre os casos de teste que atravessam os blocos e sobre a conectividade dos mesmos; então é aplicado um algoritmo para determinar quais blocos afetam e são afetados por quais outros e os casos de teste são organizados com base na travessia dos blocos conectados.

Woodward (1993) apresenta uma alternativa para a aplicação do critério Análise de Mutantes, que ele denomina *mutação firme* ("firm mutation"). Ele estabelece o conceito dessa abordagem, relacionando-a à *mutação fraca* ("weak mutation"), proposta por Howden (1982) e o critério original, isto é, a Análise de Mutantes, a qual é referenciada como *mutação forte* ("strong mutation"). O autor considera as mutações fraca e forte como os dois extremos do escopo da abordagem de mutação e a mutação firme, representando um meio termo entre elas.

A mutação fraca, proposta por Howden (1982), considera que um programa P é formado de vários componentes C, que em geral são estruturas computacionais elementares. Aplica-se em C uma mutação produzindo um componente C_i , que é o mutante de C, e requer-se que um conjunto de dados de teste construído para P execute o componente C e que, em pelo menos uma das execuções, C e C_i produzam resultados diferentes. A maior vantagem da mutação fraca sobre a forte, como comentam Offutt e Lee (1994), é que ela requer bem menos execução, pois a comparação dos resultados é feita logo após o ponto onde ocorreu a mutação. E a desvantagem é que mesmo que C e C_i produzam saídas diferentes, pode acontecer que P e P_i (o programa que contém C_i) possuam a mesma saída. Em outras palavras, isto quer dizer que casos de teste adequados à mutação fraca podem não ser adequados à mutação forte, ou ainda, com a mutação fraca não se garante que todos os erros da classe associada a um tipo de mutação sejam revelados. Offutt e Lee (1994) consideram que a mutação fraca é uma alternativa de custo bastante efetiva para o teste de unidade de programas não críticos. Para aplicações críticas, eles consideram melhor o uso da mutação forte ou então, uma combinação de ambas.

Woodward (1993) define a mutação firme como uma abordagem intermediária no teste de mutação. Definem-se, no programa, o ponto onde as alterações ocorrem e o ponto onde as saídas do programa original e do programa mutante devem ser comparadas, o qual deve ser antes do final da execução do programa. A vantagem da mutação firme em relação à mutação forte é a diminuição do custo de execução dos mutantes, uma vez que são necessárias apenas execuções parciais do programa para comparar os comportamentos dos programas mutante e original.

Existe ainda uma outra alternativa para aplicação do critério Análise de Mutantes que atua em nível de compiladores (Krauser, 1991). Em geral, os compiladores disponíveis não fornecem para os programadores e testadores nenhum suporte específico para as atividades de teste. No entanto, durante a compilação, estão disponíveis informações sintática e semântica suficientes para apoiar critérios de teste. No caso da Análise de Mutantes, segundo os autores, é possível integrar o critério diretamente dentro do compilador, permitindo que o programa possa ser modificado em nível de código executável, depois de ser compilado, não necessitando uma nova compilação. Para que seja utilizada essa estratégia, quando o programa é compilado, é preciso que sejam criados além do código objeto do programa, vários remendos ("patches"), cada um deles correspondendo ao conjunto de instruções que devem ser colocadas no programa objeto a fim de refletir uma mutação no programa fonte. Para um compilador com essa capacidade, gerar os mutantes significa incluir os remendos em pontos corretos do programa objeto. Dessa forma, evita-se que cada mutante, que tradicionalmente é gerado em nível do programa fonte, precise ser compilado separadamente, podendo assim conseguir-se uma redução no tempo necessário para a aplicação do critério Análise de Mutantes na atividade de teste.

Portanto, como pode ser observado, vários trabalhos vêm sendo conduzidos na tentativa de tornar viável a aplicação do critério Análise de Mutantes na atividade de teste. De maneira geral, todos eles visam ao mesmo objetivo de reduzir o custo e a complexidade computacional associados ao critério, diminuindo o tempo de processamento ou o número de mutantes gerados.

Mais recentemente com a disponibilidade de ferramentas de teste que apóiam a aplicação da Análise de Mutantes, (Acree, 1980; DeMillo et al, 1988; Choi et al, 1989b; Delamaro, 1993), vários estudos empíricos foram conduzidos procurando avaliar o custo, "strength" e eficácia do critério Análise de Mutantes (Horgan e Mathur, 1992; Offutt et al, 1996; Mathur e Wong, 1993; Wong et al, 1994a; Offutt, 1992), fornecendo evidências de ser este um critério atrativo para o teste de programas, dada a sua alta eficácia. A redução de custo pela abordagem "constrained mutation" não apresenta redução significativa na qualidade do teste. Por exemplo, Offutt et al (1996) determinaram um conjunto de quatro operadores para a linguagem Fortran, ou seja, gerando-se conjuntos de casos de teste adequados para esses operadores, obtém-se, em geral, uma alta cobertura para a mutação tradicional. Resultados similares são apresentados por Maldonado et al (1996) para a linguagem C, onde foram caracterizados seis operadores de mutação com alta relação com aqueles identificados por Offutt et al (1996).

Além desses trabalhos, também em decorrência da disponibilidade de ferramentas, alguns outros resultados foram apresentados. Por exemplo, em (Acree, 1980) são relatados alguns experimentos que analisam principalmente a validade do efeito de acoplamento, a capacidade de um testador em identificar mutantes equivalentes e a eficácia de cada operador de mutação, utilizando a CMS. Horgan e Mathur (1992) procuram mostrar como a Análise de Mutantes se relaciona com outros critérios de teste. Em geral, os critérios são comparados usando conjuntos de casos de teste construídos com o auxílio da Análise de Mutantes e submetendo-os aos outros critérios, ou vice-versa. Budd et al (1980) e DeMillo (1980) apresentam uma outra abordagem, onde utilizam-se programas com erros inseridos artificialmente e verificam-se se os conjuntos de casos de teste que satisfazem a Análise de Mutantes são eficazes para detectar esses erros. Já no trabalho de Acree et al (1979), explora-se outro aspecto, isto é, procura-se mostrar como a construção de alguns operadores de mutação garantem ou procuram incluir outros critérios.

O objetivo deste trabalho é explorar a adequação do uso da Análise de Mutantes para o teste e validação de especificações do aspecto comportamental de Sistemas Reativos. Iniciativas nesta direção são observadas no contexto de

protocolos de comunicação. Assim, na seção seguinte, comentam-se alguns trabalhos nessa área, observando que eles se restringem principalmente ao âmbito de Máquinas de Estados Finitos.

3.2.2 Aplicações do Critério Análise de Mutantes no Contexto de Protocolos de Comunicação

No teste e validação de protocolos de comunicação podem ser observadas algumas iniciativas do uso do critério Análise de Mutantes. O teste e validação dos protocolos são fundamentalmente baseados em um modelo de erros, que corresponde a um conjunto restrito e preestabelecido dos erros a serem explorados durante as atividades de teste. Petrenko e Bochmann (1996) comentam que a abordagem de teste baseada em um modelo de erros é muito prática no sentido de captar a intenção do testador para detectar qualquer erro de implementação, embora, no teste de conformidade de protocolos, ainda não exista um modelo de erros padrão como existe no teste de hardware.

Morell (1990) define o teste baseado em erros como sendo aquele cujo objetivo é demonstrar a ausência de erros preestabelecidos. Assim, tendo essa estratégia a finalidade de encontrar tipos específicos de erros, ela é, normalmente, uma estratégia bem sucedida pois, em geral, os programadores têm a tendência de cometer determinados erros. Com a identificação desses erros e a definição de um conjunto preestabelecido dos mesmos, a atividade de teste pode ser adequada ao ambiente de desenvolvimento do software.

Em seguida, comentam-se alguns trabalhos sobre a aplicação da Análise de Mutantes, ou variações desse critério, no teste de protocolos de comunicação. Ressalta-se que essas iniciativas foram conduzidas em época simultânea à realização deste trabalho. Dessa forma, procura-se comentar os trabalhos identificados, fazendo-se, sempre que possível, uma comparação com as definições e abordagens aqui adotadas.

Petrenko e Bochmann (1996) discutem aspetos de cobertura no teste de especificações baseadas em estados finitos. O teste é utilizado para avaliar a corretitude de uma implementação em relação aos seus requisitos de especificação. O teste ideal é o exaustivo, onde todos os possíveis valores de entrada são considerados. No entanto, como esse tipo de teste é, em geral, impraticável, o teste acaba tornando-se uma negociação entre a confiança na corretitude de uma implementação e a quantia de tempo e esforço que pode ser empregada na atividade de teste.

Os autores fazem uma comparação da cobertura no teste de software e no teste de protocolos. Eles referenciam a cobertura estrutural, relacionada ao teste caixa branca, considerando que no caso de especificações baseadas em MEFs, uma cobertura de todos os ramos equivale ao teste de todas as transições ("transition tour"), no qual muitos aspectos de controle não são testados. Esse tipo de teste, no caso de especificações de protocolos, é considerado insuficiente, ao passo que uma cobertura de 70%, no caso do teste de software, é considerada excelente.

No caso da cobertura baseada em domínios, para o teste de software ela está associada aos critérios de teste funcional, que procuram cobrir cada domínio de entrada utilizando um caso representativo do domínio e os valores extremos. Já, para o teste de protocolos, identifica-se uma diferença, pois estes são Sistemas Reativos e como tal, não possuem um único conjunto de parâmetros de entrada, mas uma seqüência infinita de eventos.

No caso de teste baseado em erros, o propósito é encontrar todos os erros relacionados ao modelo de erros definido. Essa abordagem é considerada a mais prática uma vez que ela capta todos os erros que se desejam encontrar na implementação, além do que o modelo de erros é definido com base no processo de implementação. Embora ainda não exista um modelo de erros padrão para o teste de conformidade de protocolos, vários estudos têm sido realizados em relação ao teste baseado em erros para especificações que utilizam técnicas de estados finitos.

Observa-se aqui, mais uma vez, a tendência na utilização de um modelo de erros, o qual pode ser visto como um conjunto de operadores de mutação. Com essa abordagem, existe a facilidade de selecionar-se um subconjunto de operadores de mutação, possibilitando uma forma de priorizar certos tipos de erros, além de viabilizar a introdução de outros operadores quando isso se fizer necessário. Dessa forma, pode-se adequar a atividade de teste aos recursos disponíveis e à criticalidade da aplicação.

Assim, os autores fazem uma análise dos problemas de cobertura de erros para modelos de protocolos baseados em estados finitos. O teste baseado em MEF é, normalmente, tratado como o problema de testar a implementação da MEF, ou seja, verificar se a implementação está em conformidade com a especificação. Essa relação de conformidade é muito importante na análise de cobertura do teste pois, com base nela, determinam-se quais implementações devem ser consideradas errôneas. Dada uma relação de conformidade, o conjunto de teste deve verificar se essa relação permanece entre duas MEFs para um conjunto infinito de seqüências de entrada. Por outro lado, como já foi mencionado, o conjunto de teste deve ser finito e então, a forma mais usual de limitar o número de implementações é assumir um modelo de erros.

É definida uma medida de cobertura de erros similar ao escore de mutação e então, um conjunto de teste é dito completo se para cada implementação que não esteja em conformidade com a especificação existe um teste capaz de distinguí-la, através da saída produzida. Tanto para MEFs como para MEFs Estendidas, os autores determinam, com base em alguns métodos existentes e também na teoria dos autômatos, algumas condições necessárias e suficientes para que um conjunto de teste seja considerado completo.

Os autores comentam também sobre a análise de cobertura de erros associada a um conjunto de teste, uma especificação e um modelo de erros determinados. São consideradas algumas abordagens para o cálculo da cobertura, como por exemplo:

(1) a Análise de Mutantes exaustiva, que permite determinar o valor exato da cobertura, uma vez que são gerados todos os mutantes;

(2) a simulação Monte-Carlo (Sidhu e Leung, 1989), que particiona o conjunto de máquinas mutantes em classes e, para cada classe gera aleatoriamente um número pequeno de mutantes, que são executados e que possibilitam, de acordo com seus comportamentos, estimar o valor da cobertura. Fazendo-se uma analogia com os critérios de mutação alternativa, tem-se as classes de operadores de mutação e a mutação aleatória;

(3) a análise estrutural (Yao et al, 1994a), que obtém um valor estimado do número de mutantes que são distinguidos, analisando a estrutura da máquina em relação a um conjunto de teste, permitindo então fazer uma estimativa do valor da cobertura.

Os autores concluem salientando a relevância de pesquisas que elaborem estratégias de teste combinando vários critérios de cobertura de erros para as especificações baseadas em Máquinas de Estados Finitos Estendidas e outras técnicas de descrição formal.

Probert e Guo (1991) propõem a técnica de teste E-MPT (Estelle-directed Mutation-based Protocol Testing) para o teste de especificações de protocolos escritas em Estelle, com base no teste de mutação. A técnica fornece uma maneira de construir e avaliar a completitude dos teste, além de desenvolver um conjunto de testes nas primeiras fases do processo de engenharia do protocolo, que serve de base para o teste funcional da implementação. Na fase de manutenção, onde podem surgir algumas alterações na especificação, o processo pode ser aplicado novamente para avaliar a cobertura do conjunto gerado anteriormente e também direcionar a elaboração de novos testes. O teste de protocolos tem se tornado cada vez mais crítico e várias técnicas formais de especificação como Estelle, LOTOS e SDL têm sido desenvolvidas para facilitar a especificação, como também os testes; no entanto, ainda existe uma carência de técnicas e ferramentas para validar o comportamento de tais especificações. A técnica E-MPT baseia-se em duas hipóteses:

(1) *hipótese do projetista competente* que considera que a especificação do protocolo foi realizada por um projetista competente e portanto, se não estiver correta, ela difere de uma especificação correta por pequenos erros "plausíveis", isto é, erros que são sintaticamente corretos e que pertencem a um conjunto finito de mutações nas transições;

(2) *hipótese de corretitude das partes não executáveis* que considera que a especificação está correta, exceto por suas transições executáveis, isto é, se o protocolo não estiver correto ele difere do correto por pequenos erros de transições.

Os principais passos da E-MPT são:

(1) *preparação dos operadores de mutação* que corresponde a selecionar as alternativas sintaticamente corretas, gerando um conjunto completo e finito das possíveis alterações em Estelle e, de acordo com esse conjunto, selecionar os operadores de mutação;

(2) *teste da especificação principal M* que corresponde a submeter M a um compilador Estelle que gera programas C que implementam a Máquina de Estados Finitos Estendida correspondente à especificação; adicionar o código necessário para completar os módulos gerados; compilar tais módulos e executá-los com um conjunto de casos de teste;

(3) *geração de especificações mutantes* que corresponde a gerar mutantes de M com base em (1);

(4) *teste dos mutantes* que corresponde a submeter ao compilador as especificações mutantes em Estelle, completar o código C gerado, submetê-lo ao compilador C e executar os mutantes com o conjunto de casos de teste, verificando se os mutantes permanecem vivos ou mortos;

(5) *análise* que corresponde a analisar o resultado de (4) e verificar se os mutantes vivos são equivalentes ou se o conjunto de casos de teste deve ser melhorado.

As mutações consideradas nessa técnica são classificadas em dois níveis: *mutações maiores* que testam as estruturas básicas da especificação Estelle e *mutações menores* que testam a corretitude das operações das transições. Nessa técnica são consideradas formas de organização do teste, que se refere à ordem em que o testador deve arranjar a seqüência de mutações do módulo em teste, e também mecanismos para aumentar a velocidade do teste, o que está relacionado com aspectos de eficácia e eficiência, como por exemplo o Teste de Mutação Ordenado, proposto por Duncan e Robson (1990), o qual propõe uma ordenação dos casos de teste e dos operadores de mutação. Além desses dois mecanismos, na técnica E-MPT é proposta também uma ordenação das aplicações mutantes.

Outra forma de aplicação da técnica, além de testar especificações em Estelle, é para avaliação de estratégias de projeto de testes, comparando sua eficácia.

Observa-se que a abordagem adotada por Probert e Guo (1991), é bastante similar à abordagem adotada neste trabalho, onde traça-se um paralelo entre as hipóteses do critério Análise de Mutantes entre os níveis de programa e de especificação. A hipótese do projetista competente corresponde à hipótese do especificador competente, neste trabalho, como será descrito na Seção 3.3. Por outro lado, notam-se duas diferenças: uma delas é que neste trabalho, os operadores de mutação estão definidos em nível da Máquina de Estados Finitos, de forma matemática, caracterizando precisamente o tipo de erro que eles modelam, enquanto que na abordagem apresentada pelos autores, são definidos cinco operadores, em termos da especificação Estelle, de maneira menos formal. Dentre esses operadores, dois deles podem ser relacionados aos operadores *evento trocado* e *destino trocado*, definidos neste trabalho. A outra diferença, decorrente da anterior, corresponde ao aspecto de simulação, pois neste trabalho, as mutações são realizadas no próprio nível da MEF e a simulação dos mutantes também é feita nesse nível. Já na abordagem de Probert e Guo, os operadores de mutação atuam na especificação Estelle gerando um mutante, com base no qual é gerado um código em C, que por sua vez é completado pelo usuário para tornar-se equivalente à especificação mutante, em Estelle. Após esse processo, é que os casos de teste são aplicados, em nível do programa C. Ressalta-se que na execução desses passos, principalmente com a interferência do usuário para a complementação do código em C, novos erros podem estar sendo introduzidos.

Chung e Sidhu (1991) propõem uma técnica para geração de seqüências de teste probabilísticas para o teste de conformidade de protocolos, e avaliam a cobertura dessas seqüências através de um procedimento que se assemelha aos passos da Análise de Mutantes. Um padrão de protocolo pode ter vários tipos de implementação diferentes e, para assegurar que diferentes implementações estejam aptas a operar umas com as outras é necessário que se teste a conformidade de uma implementação em relação a um padrão. Para o teste de conformidade pode-se usar uma seqüência gerada a partir da especificação formal, aplicando-a na implementação em teste.

No entanto, como os protocolos são sistemas complexos, normalmente não é possível gerar seqüências que cubram todos os comportamentos dos mesmos. Assim, os autores propõem uma outra abordagem, denominada "P-Method" que cobre os subconjuntos mais prováveis de comportamento através das seqüências geradas. Em geral, os protocolos são especificados através de Máquinas de Estados Finitos, uma para cada entidade.

Embora existam vários métodos de geração de seqüências de teste para MEFs, estas são geradas para uma única entidade, sem considerar a outra entidade da interação, o que significa que a seqüência testa uma MEF em relação ao seu padrão, ignorando muitos caminhos quando considerada a outra entidade. Por outro lado, testar todas as possíveis interações existentes entre todas as entidades também não é viável. Assim, os autores atribuem probabilidades às transições de acordo com as freqüências que elas possivelmente ocorram e fornecem um guia para explorar primeiramente as partes mais prováveis.

Para verificar a cobertura dessas seqüências de teste probabilísticas geram-se, randomicamente, através do procedimento da simulação Monte-Carlo (Sidhu e Leung, 1989) com algumas modificações, máquinas erradas que possuem pequenas diferenças em relação à máquina especificada. Essas máquinas erradas representam dez classes de erros relacionadas a modificações nos estados destinos e/ou saídas de um ou mais arcos da máquina especificada. Esses erros estão relacionados com erros de transferência e de operação, de acordo com as classes de erros propostas por Chow (1978). Observa-se que os erros definidos em cada uma dessas dez classes, além de retratarem um erro de transferência e/ou operação, retratam também o aspecto de geração de mutantes de ordem k (para k = 1, 2, 3, 4 e 5). Por exemplo, uma dessas dez classes corresponde a alterar o estado destino de três arcos e a saída de outros dois arcos da MEF.

Como a seqüência probabilística pode cobrir apenas um subconjunto do protocolo, definiram-se duas medidas de cobertura:

(1) *estimativa forte*, que calcula a cobertura considerando máquinas geradas pela modificação de qualquer arco da especificação;

(2) *estimativa fraca*, que calcula a cobertura considerando máquinas geradas pela modificação apenas daqueles arcos da especificação que são cobertos pela seqüência de teste.

Os passos principais para estimativa da cobertura de uma seqüência probabilística são:

(1) realizar a especificação da MEF;
(2) definir a seqüência a ser usada;
(3) especificar o conjunto de arcos que podem sofrer as modificações;
(4) gerar uma máquina errada para cada uma das dez classes de erros;
(5) aplicar a seqüência a cada uma das máquinas geradas;
(6) analisar se as máquinas que passam pela seqüência estão realmente em conformidade com a máquina especificada.

É também estabelecida uma medida de cobertura, *fc*, dada pela seguinte relação: fc = (número de máquinas geradas randomicamente - número de máquinas que têm o mesmo comportamento da máquina original para a seqüência de teste) / (número de máquinas geradas randomicamente - número de máquinas equivalentes).

Como pode ser observado, o procedimento utilizado no "P-method" assemelha-se à idéia básica do critério Análise de Mutantes no que diz respeito à geração das máquinas com pequenas alterações, aos passos utilizados para avaliação da seqüência probabilística e também ao cálculo da cobertura dessa seqüência.

A proposta de Chung e Sidhu (1991) também assemelha-se à proposta feita neste trabalho, mas algumas diferenças também podem ser observadas. Os erros modelados pelos autores restringem-se a alteração do estado destino e da saída gerada pelas transições e combinações desses dois aspectos. Além disso, essas classes não estão formalmente definidas, supondo-se que as alterações também sejam feitas de forma aleatória, assim como é o número de máquinas mutantes que são geradas para cada uma das classes de erros. Outro aspecto que diferencia deste trabalho é que a maioria das classes de erros correspondem à geração de máquinas com várias alterações simultâneas, isto é, geração de

mutantes de ordem k. Esse aspecto pretende-se tratar em trabalhos futuros, para avaliar-se o efeito de acoplamento em nível de especificações, uma vez que, em nível de programas esse efeito já foi constatado em vários experimentos empíricos.

Bochmann e Petrenko (1994) explicam que como os protocolos de comunicação são, em geral, de natureza reativa, as linguagens de especificação para Sistemas Reativos são bastante apropriadas para descrevê-los. Assim, torna-se compreensível a ampla utilização de Máquinas de Estados Finitos para a especificação de protocolos e, conseqüentemente, que a maioria dos trabalhos relacionados ao teste de protocolos sejam baseados em MEFs. Salientam que assim como no teste de software em geral, um aspecto muito importante é obter-se uma avaliação da cobertura dos erros para um conjunto de teste. Os métodos que têm sido propostos são essencialmente variações da Análise de Mutantes, como é o caso do trabalho de Chung e Sidhu (1991), citado anteriormente. Para determinar a cobertura, é necessário determinar quais erros de uma implementação devem ser cobertos, ou seja, quais mutações devem ser consideradas, de forma que um conjunto de implementações com comportamento errôneo deva ser detectado pelo conjunto de teste.

No caso de especificações baseadas em MEFs determinísticas e completamente especificadas, normalmente os tipos de mutações considerados são *erros de saída* ("output faults" - a saída de uma transição é incorreta) ou *erros de transferência* ("transfer faults" - o próximo estado de uma transição é incorreto). Erros de estados extras não são considerados, uma vez suposto que é realizada uma implementação direta da especificação. Muitos métodos para derivar seqüências de teste para esse tipo de MEF são propostos (Chow, 1978; Fujiwara et al, 1991) mas no entanto, em geral, as especificações de protocolos não satisfazem as suposições iniciais para a aplicação dos mesmos.

No caso de especificações baseadas em MEFs parcialmente especificadas, elas são transformadas para completamente especificadas tornando as transições:

(1) *definidas implicitamente* - fazendo com que a transição retorne para o próprio estado ou seja direcionada para um estado de erro, sem associar saída à mesma;

(2) *indefinidas por "default"* - fazendo com que a transição seja direcionada para qualquer estado, com qualquer saída ou

(3) *esquecidas* - fazendo com que não sejam submetidas determinadas entradas para determinados estados.

Tanto no experimento realizado manualmente, apresentado no Capítulo 4, como também no simulador do ambiente StatSim, utilizado na implementação da ferramenta Proteum-RS/FSM, as MEFs parcialmente especificadas são transformadas em completamente especificadas, tomando as transições definidas implicitamente.

No caso de especificações baseadas em MEFs não determinísticas, ocorre que um determinado teste pode gerar diferentes saídas e então, não é suficiente executar o mesmo teste uma única vez, devendo este ser repetido até que sejam obtidas todas as possíveis observações de saídas. A geração de testes para esse tipo de máquina é uma área ativa de pesquisa e diversos métodos têm sido propostos, alguns deles baseados em heurísticas e outros, em modelos de erros (Yevtushenko et al, 1991; Petrenko et al, 1993; Petrenko, 1991; Luo et al, 1994).

No caso de Máquinas de Estados Finitos Estendidas, as especificações possuem também variáveis adicionais e ações e predicados associados às transições. Em geral, a notação usada nos predicados e ações está fortemente ligada a uma linguagem de programação. Assim, o teste de implementações baseadas em MEFs Estendidas está bastante relacionado ao teste de software, gerando inclusive questões como a decisão sobre a executabilidade de um determinado ramo do programa. Existem várias propostas para utilização da análise de fluxo de dados para a seleção de casos de teste para essas especificações. A técnica de mutação também tem sido utilizada para gerar seqüências de teste para MEFs Estendidas; alguns mutantes são construídos e então os comportamentos das máquinas são comparados a fim de gerar-se uma seqüência de teste (Probert e Guo, 1991; Wang e Liu, 1993).

Em resumo, comentam os autores, que os métodos que vêm sendo desenvolvidos para o teste de protocolos possuem um grande escopo de

aplicação, como por exemplo, Sistemas Reativos, sistemas de controle, e também o teste funcional de circuitos seqüenciais.

Outro trabalho também na área de teste de conformidade de protocolos foi apresentado por Wang e Liu (1993), considerando que ambas as especificações do protocolo e do modelo errado, isto é, do mutante, estão descritas através de Máquinas de Estados Finitos Estendidas. Um dos pontos mais importantes na área de teste de conformidade de protocolos é a geração de casos de teste. Existem alguns métodos para a geração de casos de teste, mas a maioria deles está baseada em MEFs. Outros métodos de geração de casos de teste estão baseados em Estelle (Budkowski e Dembinski, 1987) e LOTOS (Bolognesi e Brinksma, 1987), mas pelo fato desses modelos usualmente utilizarem memória, além de testar o fluxo de controle, torna-se necessário o teste dos aspectos relacionados aos dados, o que dificulta muito a geração dos casos de teste. Os autores comentam que um caso de teste é menos significativo quando não se sabe seu propósito. O propósito do caso de teste indica o tipo de erro que esse tenta detectar e ele deve ser estabelecido antes que qualquer caso de teste significativo seja construído. No método proposto, em vez dos modelos de falhas (como são chamados os erros) serem fixados, a definição desses modelos é deixada para os usuários, que podem definir os erros que eles consideram mais importantes. Os casos de teste são gerados comparando as diferenças entre a especificação e o modelo errado. É definida uma entrada inicial, que é aplicada tanto na especificação como no mutante e essa entrada vai sendo expandida até que seja alcançada uma situação onde a especificação e o mutante geram um comportamento diferente.

Observa-se nesse trabalho, que a adequação do modelo de falhas às necessidades dos usuários é uma forma de aplicar-se a "constrained mutation" apresentada em (Wong et al, 1994b), e utilizada, neste trabalho, através dos critérios de mutação alternativa propostos. A escolha dos erros a serem detectados, ou seja, a escolha dos operadores de mutação a serem aplicados é uma forma de adequar as atividades de teste às características próprias do ambiente, além de se conseguir com isso uma redução do custo, esperando que não se perca na eficácia dessas atividades.

Quanto ao desenvolvimento de ferramentas, alguns trabalhos vêm sendo conduzidos no contexto do teste e validação de protocolos de comunicação, como é o caso do trabalho apresentado por Tan et al (1996). Nesse trabalho, os autores descrevem uma ferramenta, denominada *TAG* (Test Automatic Generation), que gera casos de teste automaticamente para especificações baseadas em MEFs, implementando uma abordagem de identificação das transições. Essa ferramenta apóia o teste de conformidade de protocolos, derivando um conjunto de teste para MEFs determinísticas e parcialmente especificadas.

Embora não se trate de uma aplicação diretamente relacionada ao teste de protocolos de comunicação, em (Weyuker et al, 1994), como é apresentado em seguida, os autores fazem uma proposta de operadores de mutação para a validação de expressões booleanas, os quais podem ser aplicados no teste de Máquinas de Estados Finitos Estendidas. Com base nessa proposta foram definidos alguns dos operadores de mutação apresentados no Capítulo 6, no contexto do teste de especificações baseadas em Statecharts.

Weyuker et al (1994) propõem um método básico de seleção de casos de teste para sistemas que podem ser descritos através de expressões booleanas, e também algumas variantes do mesmo. A estratégia básica, chamada de estratégia do impacto significativo, baseia-se na idéia de que para testar uma fórmula booleana, devem-se selecionar casos de teste que sejam capazes de demonstrar, quando possível, que cada literal da fórmula possui um impacto significativo no valor da função. Isto acontece quando alterando-se o valor do literal obtém-se um valor diferente para a fórmula. Dessa maneira, se um conjunto de casos de teste não é capaz de demonstrar o impacto significativo para cada literal, um literal pode ser negado erroneamente, gerando um erro não revelado, pois o efeito não será sentido no valor da expressão. Definido o método básico, os autores apresentam uma avaliação analítica para determinar o tipo de erro que esse método consegue ou não revelar. Com essa análise, verificou-se que determinados erros não eram identificados pela estratégia básica, e então foram definidas seis estratégias variantes da estratégia inicial. Então, para determinar a capacidade de revelar erros de cada uma das estratégias, os autores utilizaram o critério Análise de Mutantes, que é também um critério de adequação de casos de teste. Para utilização desse critério foram definidos os operadores de mutação que modelam

os tipos de erros que se pretende revelar com os testes. Os autores determinaram cinco operadores de mutação:
(1) erro de negação de variável;
(2) erro de negação de expressão;
(3) erro de referência a variável;
(4) erro de referência a operador e
(5) erro na associatividade dos termos.

Várias seqüências de teste foram geradas para várias expressões booleanas e, aplicando-se o critério Análise de Mutantes e calculando-se o escore de mutação em cada caso, os autores comprovam o benefício alcançado ao utilizarem-se as estratégias propostas em vez de utilizarem-se casos de teste gerados aleatoriamente.

No Capítulo 6, os operadores de mutação definidos por Weyuker et al (1994), serão tratados com maior detalhe, pois eles são considerados na definição dos operadores de mutação para Máquinas de Estados Finitos Estendidas, propostos neste trabalho.

De acordo com os trabalhos comentados anteriormente, o que se observa é que a Análise de Mutantes tem sido aplicada em vários contextos e que, em nível de especificação, a utilização desse critério concentra-se principalmente no teste de conformidade de protocolos de comunicação, mais particularmente, através de especificações baseadas em Máquinas de Estados Finitos. Neste trabalho, além da aplicação da Análise de Mutantes em MEFs propõe-se também sua aplicação em Máquinas de Estados Finitos Estendidas, Redes de Petri e Statecharts. Pelo fato da técnica Statecharts ser uma extensão de Máquinas de Estados Finitos e, conseqüentemente, por abranger os conceitos relativos a MEFs Estendidas, os operadores de mutação para esta última técnica são apresentados no Capítulo 6, onde aborda-se a aplicação da Análise de Mutantes em Statecharts. Para a técnica Redes de Petri, os resultados da aplicação do critério nessa técnica, assim como a definição dos operadores de mutação são apresentados no Capítulo 5.

3.3 O Critério Análise de Mutantes Aplicado em Nível de Especificação de Sistemas

Como foi colocado inicialmente, o critério Análise de Mutantes é, originalmente, um critério utilizado para o teste de programas. Este trabalho visa sistematizar e apoiar através de ferramentas a aplicação do critério Análise de Mutantes na validação do aspecto comportamental de Sistemas Reativos. Um ponto importante nesta perspectiva é que procura-se adequar a terminologia e atividades relativas ao uso da Análise de Mutantes para o nível de especificação. Para tanto, como é apresentado em (Fabbri et al, 1993; Fabbri et al, 1994b), traçou-se um paralelo entre as hipóteses básicas do critério, em nível de programas, para o nível de especificações.

Para o projeto dos operadores de mutação considera-se a ***hipótese do especificador competente***: dada uma especificação S, gera-se um conjunto de mutantes de S, $\Phi(S)$, com base em um conjunto de operadores, cuja definição é um ponto fundamental para a aplicação desta técnica, e diz-se que um conjunto de teste T é adequado para S em relação a Φ se para cada especificação Z pertencente a Φ, ou Z é equivalente a S ou Z difere de S em pelo menos um ponto de T.

Para a aplicação dos operadores de mutação em nível de programas, considera-se o efeito de acoplamento; na realidade, esse conceito deve ser "validado neste novo contexto", através da aplicação do critério em vários exemplos e da observação dos tipos de erros que são detectados. Em nível de programas, de acordo com esse conceito, é suficiente que sejam gerados apenas mutantes de ordem 1 (um), pois erros mais complexos são detectados através de casos de teste elaborados com a finalidade de detectar erros mais simples.

Estabelecido esse paralelo entre os níveis de programa e de especificação, propõe-se a aplicação do critério Análise de Mutantes na validação de especificações baseadas em Máquinas de Estado Finito (Gill, 1962), Redes de Petri (Peterson, 1977) e Statecharts (Harel, 1987a), como será apresentado nos capítulos subseqüentes.

Embora algumas aplicações justifiquem o alto custo das atividades de VV&T, o critério Análise de Mutantes pode ser impraticável em situações reais, como tem sido observado no contexto do teste de programas. Devido ao grande número de mutantes gerados e, conseqüentemente, o alto custo e o tempo de processamento que são decorrentes desse fator, várias alternativas têm sido propostas para a aplicação do critério em nível de programas. Uma dessas alternativas se resume nos trabalhos de Wong (Wong, 1993; Wong et al, 1994a; Wong et al, 1994b), relativos aos critérios de mutação alternativa ("constrained mutation"), que em nível de programas conseguem reduzir o custo e o tempo de processamento, sem perder a eficácia característica desse critério.

Assim, com base nos resultados apresentados por Wong, em trabalhos citados anteriormente, identificam-se claramente as contribuições alcançadas no teste de unidade, pela utilização dos critérios alternativos da Análise de Mutantes. Como a geração de um grande número de mutantes é uma característica inerente ao critério, justifica-se explorar, também em nível de especificações, formas alternativas para sua utilização. Inspirando-se, portanto, no trabalho de Wong, propõe-se neste trabalho a definição e utilização de critérios alternativos para a utilização da Análise de Mutantes em nível de especificações, como é apresentado na próxima seção.

Salienta-se ainda, retomando alguns pontos da Seção 3.2.2, que a definição de um modelo de erros é tratada como uma forma natural e bastante prática no teste de conformidade de protocolos e que um modelo de erros corresponde a determinadas mutações que se desejam detectar nas especificações dos mesmos. Além disso, pode-se caracterizar inclusive o conceito de "constrained mutation" nos trabalhos citados pois a utilização de um modelo de erros permite que a atividade de teste se torne bastante maleável, viabilizando uma "negociação" entre os itens orçamento e criticalidade da aplicação. Nesse sentido, os usuários podem escolher e/ou definir tipos de erros para o modelo de erros utilizado, de forma a adequá-lo às necessidades e características do ambiente.

3.3.1 Mutação Alternativa: Mutação Restrita e Mutação Aleatória

Apresentam-se nesta seção duas alternativas para o critério Análise de Mutantes, a exemplo do que foi proposto por Wong et al (1994b), em nível de programas. Embora esses estudos tenham sido apresentados num escopo limitado, seus resultados motivam uma investigação mais sistemática que leve a conclusões mais seguras. Mesmo assim, pelas evidências apresentadas (Wong et al, 1994b), definem-se e exploram-se neste trabalho as seguintes formas alternativas de mutação em nível de especificação:

- **Critério de Mutação Aleatória:** examina uma pequena porcentagem de mutantes selecionados aleatoriamente de cada tipo de mutantes e ignoram-se os demais. Esse tipo de mutação é referenciado como mutação n%; no caso, utilizou-se 10% de cada tipo de mutantes. Portanto, este critério é referenciado como Mutação 10%. O número de mutantes examinados, para cada tipo de mutantes, é a parte inteira de n%, aplicando-se o arredondamento caso a parte decimal seja maior ou igual a 0.5. Se a porcentagem é menor que 1 examina-se um mutante. Assim, o número total de mutantes examinados na mutação n% é de $(n + \delta)$ % , onde δ é um número bastante pequeno.

- **Critério de Mutação Restrita:** examina apenas alguns tipos específicos de mutantes e ignoram-se os demais. Deve-se observar que a seleção dos operadores de mutação é um ponto crucial e pode levar a uma redução significativa no custo de execução sem sacrificar a força e a capacidade de detecção de falhas do teste de mutação, como é observado por Wong et al (1994b).

A fim de comparar esses critérios, utilizam-se neste trabalho, as mesmas métricas propostas por Wong et al (1994b), relacionadas ao custo e ao "strength" (dificuldade de satisfação do critério):

Custo:

$$C_1 = \left(1 - \frac{\textit{tamanho médio dos conjuntos de teste adequados com relação à mutação alternativa}}{\textit{tamanho médio dos conjuntos de teste adequados à mutação}}\right) \times 100\% \quad (2)$$

$$C_2 = \left(1 - \frac{\textit{número total de mutantes examinados usando mutação alternativa}}{\textit{número total de mutantes examinados na mutação}}\right) \times 100\% \quad (3)$$

"Strength":

$$C_3 = \left(\frac{\textit{número de mutantes distinguidos por um conjunto de teste}}{\textit{número total de mutantes não equivalentes}}\right) \times 100\% \quad (4)$$

Os critérios de mutação alternativa definidos nesta seção, assim como as métricas para avaliá-los em relação ao critério Análise de Mutantes, foram aplicados no contexto de Máquinas de Estados Finitos e de Redes de Petri. Os resultados obtidos dessa aplicação são apresentados e comentados nos Capítulos 4 e 5, para cada uma dessas técnicas, respectivamente.

3.3.2 Convenção das Siglas Atribuídas aos Operadores de Mutação

Os operadores de mutação definidos neste trabalho possuem associados a eles siglas mnemônicas que têm por objetivo representá-los principalmente na ferramenta Proteum-RS/FSM, que apóia a aplicação da Análise de Mutantes em Máquinas de Estados Finitos e nas versões para Redes de Petri e Statecharts, que serão implementadas futuramente. A determinação dessas siglas foi inspirada no trabalho de Agrawal et al (1989), onde são definidos os operadores de mutação para a linguagem de programação C e também, onde é proposto um sistema de convenção de siglas para tais operadores.

As siglas utilizadas identificam, quando necessário, a técnica de especificação para a qual o operador é definido, através de um prefixo, que pode ser: FSM - para Máquinas de Estados Finitos (Finite State Machines),
PN - para Redes de Petri (Petri Nets) e
ST - para Statecharts

Após o prefixo, a sigla é composta por partes que abreviam a ação realizada pelo operador de mutação, sendo que a parte inicial identifica a estrutura da técnica de especificação (Máquina de Estados Finitos, Rede de Petri ou Statecharts) que está sendo alterada pelo operador. Em seguida, apresentam-se alguns exemplos de siglas de acordo com a convenção definida:

1) **FSM-TraDesStaAlt** - corresponde ao operador "destino trocado"
FSM: prefixo que identifica a técnica Máquina de Estados Finitos
Tra: referencia a estrutura que é alterada - "transition"
DesSta: abrevia "Destination State"
Alt: abrevia "Alteration"

2) **PN-InpDel** - corresponde ao operador "input faltando"
PN: prefixo que identifica a técnica Rede de Petri
Inp: referencia a estrutura que é alterada - "input"
Del: abrevia "deletion"

3) **ST-TraExpDel** - corresponde ao operador "exclusão de expressão"
ST: prefixo que identifica a técnica Statecharts
Tra: referencia a estrutura que é alterada - "transition"
Exp: abrevia "expression"
Del: abrevia "deletion"

4) **ST-TraHistDel** - corresponde ao operador "desassocia história da transição"
ST: prefixo que identifica a técnica Statecharts
Tra: referencia a estrutura que é alterada - "transition"
Hist: abrevia "history"
Del: abrevia "deletion"

Nas Tabelas a seguir apresentam-se os operadores definidos neste trabalho, com as respectivas siglas, para as técnicas Máquinas de Estados Finitos (Tabela 3.1), Redes de Petri (Tabela 3.2) e Statecharts (Tabela 3.3a, b e c). As definições desses operadores são apresentadas nos Capítulos 4, 5 e 6, respectivamente, onde trata-se a aplicação da Análise de Mutantes em cada uma das três técnicas.

Tabela 3.1 - Operadores de Mutação para Máquinas de Estados Finitos

Sigla	*Operador*
TraIniStaAlt	alteração do estado inicial Transition Initial State Alteration
TraArcDel	arco faltando Transition Arc Deletion
TraEveDel	evento faltando Transition Event Deletion
TraEveIns	evento extra Transition Event Insertion
TraEveAlt	evento trocado Transition Event Alteration
TraDesStaAlt	destino trocado Transition Destination State Alteration
OutDel	saída faltando Output Deletion
OutAlt	saída trocada Output Alteration
StaDel	estado faltando State Deletion
StaInsOutSep	isola saída relevante State Insertion Output Separation
StaInsTraSep	desmembra transição com mais de um evento State Insertion Transition Separation
StaInsTraOutSep	separa eventos com mesma saída de uma transição State Insertion Transition Output Separation
StaInsTraComb	combina transições State Insertion Transition Combination

Tabela 3.2 - Operadores de Mutação para Redes de Petri

Sigla	*Operador*
InpDel	input faltando Input Deletion
InpIns	input extra Input Insertion
InpShift	input deslocado Input Shifted
InpAlt	input trocado Input Alteration
OutDel	output faltando Output Deletion
OutIns	output extra Output Insertion
OutShift	output deslocado Output Shifted
OutAlt	output trocado Output Alteration
MarkAlt	alteração da marcação inicial Mark Alteration

Tabela 3.3a - Operadores de Mutação para Statecharts que Abordam Aspectos Relacionados a Máquinas de Estados Finitos

Sigla	*Operador*
TraDefStaDel	alteração do estado default Transition Default State Alteration
TraArcDel	arco faltando Transition Arc Deletion
TraEveDel	evento faltando Transition Event Deletion
TraEveIns	evento extra Transition Event Insertion
TraEveAlt	evento trocado Transition Event Alteration
TraDesStaAlt	destino trocado Transition Destination State Alteration
TraActDel	ação faltando Transition Action Deletion
TraActAlt	ação trocada Transition Action Alteration
StaDel	estado faltando State Deletion

Tabela 3.3b - Operadores de Mutação para Statecharts que Abordam Aspectos Relacionados a Máquinas de Estados Finitos Estendidas

Sigla	*Operador*
TraExpDel	exclusão de expressão Transition Expression Deletion
TraNegBoolExp	negação de expressão booleana Transition Negation Boolean Expression
TraAssRelAlt	troca associatividade dos termos Transition Associative Relation Alteration
TraAritOperArit	troca operador aritmético por operador aritmético Transition Aritmetic Operator by Aritmetic Operator
TraRelOperRel	troca operador relacional por operador relacional Transition Relational Operator by Relational Operator
TraLogOperLog	troca operador lógico por operador lógico Transition Logical Operator by Logical Operator
TraNegLogExp	negação Lógica Transition Negation Logical Expression
TraVarAltVar	troca variável por variável Transition Variable Alteration by Variable
TraVarAltCons	troca variável por constante Transition Variable Alteration by Constant
TraConsAltCons	troca de constantes por constantes requeridas Transition Constant Alteration
TraConsAltVar	troca constante por variável escalar Transition Constant Alteration by Variable

Tabela 3.3c - Operadores de Mutação para Statecharts que Abordam Aspectos Intrínsecos a essa Técnica

Sigla	*Operador*
HistDelTra	desassocia história da transição History Deletion from Transition
HistTraAltTra	troca transição associada ao símbolo de história History Transition Alteration Transition
HistDelSta	exclui história do estado History Deletion from State
HistHH*	troca h por h* History H by H*
HistH*H	troca h* por h History H* by H
HisInsHSta	associa h ao estado History Insertion H to State
HisInsH*Sta	associa h* ao estado History Insertion H* to State
TraCondInDel	exclui condição in(s) Transition Condition In Deletion
TraCondInStaAlt	troca estado da condição in(s) Transition Condition In State Alteration
TraCondNYDel	exclui condição not-yet(e) Transition Condition NY Deletion
TraCondNYEveAlt	troca evento da condição not-yet(e) Transition Condition NY Event Alteration
TraEveExDel	exclui evento exit(s) Transition Event Ex Deletion
TraEveExStaAlt	troca estado do evento exit(s) Transition Event Ex State Alteration
TraEveEnDel	exclui evento entered(s) Transition Event En Deletion
TraEveEnStaAlt	troca estado do evento entered(s) Transition Event En State Alteration
TraBrOrigAlt	altera transição origem do broadcasting Transition Broadcasting Origin Alteration
TraBrDestAlt	altera transição destino do broadcasting Transition Broadcasting Destination Alteration

3.4 Considerações Finais

Neste capítulo apresentou-se o conceito do critério Análise de Mutantes, um dos critérios da técnica de teste baseada em erros. A Análise de Mutantes tem se mostrado bastante eficaz na detecção de erros, mas pelo fato de gerar um grande número de mutantes, sua aplicação se torna restritiva devido ao tempo e custo de processamento.

Com o constante desenvolvimento do hardware e de novas arquiteturas de computadores, grande ênfase tem sido dada ao desenvolvimento de ferramentas que apóiem a utilização do critério. Além disso, várias pesquisas têm sido conduzidas no sentido de definir formas alternativas para a Análise de Mutantes, que façam com que a aplicação do critério se torne menos dispendiosa, sem perder em sua eficácia. Nesse sentido, foram comentados alguns trabalhos que têm por objetivo tornar a aplicação da Análise de Mutantes mais eficiente, a fim de torná-la viável em termos de custo.

Também foram comentados alguns trabalhos que apresentam pesquisas sobre a aplicação da Análise de Mutantes em contextos diferentes daquele para o qual o critério foi criado, isto é, o teste de programas. Observou-se que a maioria desses trabalhos está concentrada no teste de conformidade de protocolos de comunicação, quando estes são especificados através de Máquinas de Estados Finitos.

Como, no contexto desta pesquisa, o critério Análise de Mutantes está sendo utilizado no teste de especificações do aspecto comportamental de Sistemas Reativos, estabeleceu-se um paralelo, entre os níveis de programa e de especificação, relativo às hipóteses básicas sobre as quais o critério foi construído. Dessa forma, propõe-se, nos próximos capítulos, a aplicação do critério Análise de Mutantes no teste de especificações baseadas em Máquinas de Estados Finitos, no Capítulo 4, Redes de Petri, no Capítulo 5 e Statecharts, no Capítulo 6, no qual apresentam-se também operadores de mutação que exploram aspectos específicos de Máquinas de Estados Finitos Estendidas. Todos os operadores definidos no trabalho foram relacionados neste capítulo a fim de exemplificar a convenção utilizada para as siglas atribuídas a esses operadores.

CAPÍTULO 4

ANÁLISE DE MUTANTES APLICADA EM MÁQUINAS DE ESTADOS FINITOS

4.1 Considerações Iniciais

Segundo Gill (1962), a teoria de Máquinas de Estados Finitos (MEF) trata de modelos matemáticos que servem como aproximação de fenômenos físicos ou abstratos. A relevância da teoria das MEFs é que seus modelos são aplicáveis, praticamente, a todo campo de investigação e não se limitam a uma determinada área científica. No desenvolvimento de software, essa técnica é bastante utilizada na especificação do aspecto comportamental de Sistemas Reativos, em particular na especificação de protocolos de comunicação, como foi mencionado no Capítulo 3.

No Capítulo 2, foram comentados alguns métodos de geração de seqüências de teste para MEFs, como por exemplo, os métodos UIO, DS, TT, W e Wp, mais referenciados na literatura (Sabnani e Dahbura, 1988; Gonenc, 1970; Naito e Tsunoyama, 1981; Chow, 1978; Fujiwara et al, 1991). No entanto, a maioria desses métodos exige que as MEFs satisfaçam uma série de requisitos para que, quando aplicados, possibilitem inferir características e propriedades das mesmas. Por exemplo, os métodos W e Wp exigem que a MEF seja minimal e no caso dos métodos UIO e DS, as seqüências geradas por eles nem sempre existem, mesmo que a MEF satisfaça as propriedades de minimalidade, especificação completa e conexão forte (Fujiwara et al, 1991).

Na prática, as MEFs geralmente não satisfazem os requisitos iniciais para a aplicação desses métodos de geração de seqüências de teste. Se consideradas as Máquinas de Estados Finitos Estendidas, que possuem adicionalmente o uso de variáveis e condições, a atividade de teste se torna ainda mais difícil e pouco se encontra na literatura para o teste de especificações baseadas nessa técnica. Bochmann e Petrenko (1994) mencionam alguns trabalhos que vêm sendo conduzidos no sentido de utilizarem-se critérios de teste estrutural para o teste das MEFs Estendidas. No caso da técnica Statecharts, que é uma extensão das MEFs, não se tem conhecimento de métodos de geração de seqüências de teste. As especificações baseadas nessa técnica são testadas, mais freqüentemente, através de simulação, como as execuções interativa, em batch, programada e exaustiva (Harel, 1992).

Assim, com o objetivo de explorar formas alternativas de teste para especificações do aspecto comportamental de Sistemas Reativos baseadas nessas técnicas, aplicou-se, através de um experimento, o critério Análise de Mutantes na validação de Máquinas de Estados Finitos, para verificar o aspecto complementar do mesmo em relação aos demais métodos de geração de seqüências de teste existentes. Esse experimento foi realizado de forma manual, utilizando-se duas seqüências de teste – uma gerada pelo método TT e outra, pelo método W – para a execução dos mutantes de uma MEF que corresponde à especificação de um protocolo de comunicação, área onde essa técnica é fortemente utilizada e também onde foram encontradas algumas aplicações desse critério de teste. Nesse experimento ainda foram aplicados os critérios alternativos da Análise de Mutantes mencionados no Capítulo 3, isto é, a Mutação Restrita e a Mutação Aleatória, com o objetivo de avaliar essas formas de mutação em nível de especificações. Ressalta-se que com a aplicação de critérios de teste torna-se possível suprir a necessidade de quantificar-se a atividade de teste, com o uso, por exemplo, de análise de cobertura, que indica quanto dos requisitos de um critério foi satisfeito no teste em questão.

Salientou-se no Capítulo 3, que o ponto chave do critério Análise de Mutantes é a definição do conjunto de operadores de mutação. Portanto, assim como para as outras técnicas abordadas neste trabalho, fez-se necessária a definição desse conjunto de operadores que, para o caso de Máquinas de Estados

Finitos, foi inspirada na classificação de erros para MEFs proposta por Chow (1978).

Considerando-se também o objetivo desta pesquisa de definir uma estratégia de teste para especificações baseadas em Statecharts e, sendo a técnica Statecharts uma extensão de MEF, a aplicação da Análise de Mutantes em MEF foi um primeiro passo em direção a esse objetivo. Como será observado no Capítulo 6, um componente básico em uma especificação Statecharts corresponde a uma MEF e portanto, os operadores de mutação e os resultados aqui apresentados aplicam-se diretamente no contexto dessa outra técnica.

O capítulo está organizado da seguinte maneira: na Seção 4.2, conceitua-se, de forma resumida, a técnica Máquinas de Estados Finitos. Na Seção 4.3 apresentam-se os operadores de mutação para MEFs propostos neste trabalho. Na Seção 4.4 exemplifica-se a aplicação do critério Análise de Mutantes, com base no conjunto de operadores de mutação proposto, através de uma MEF que especifica um protocolo de comunicação. Na Seção 4.5 apresenta-se a aplicação dos dois critérios de mutação alternativa mencionados no capítulo anterior e na Seção 4.6 apresentam-se as considerações finais.

4.2 Máquinas de Estados Finitos

Nesta seção apresenta-se uma definição mais formal das Máquinas de Estados Finitos e retomam-se algumas propriedades que são utilizadas neste capítulo.

Segundo Fujiwara et al (1991), uma Máquina de Estados Finitos M, determinística, pode ser representada por uma quíntupla (X,E,Y,T,O), onde:

X : conjunto de entradas x

E : conjunto de estados, com S_0 estado inicial

Y : conjunto de saídas y

T : função de transição, $X \times E \rightarrow E$

O : função de saída, $X \times E \rightarrow Y$

A notação S_i — x/y → S_j indica que a Máquina de Estados Finitos M, no estado S_i, responde com uma saída *y* e transiciona para o estado S_j quando a entrada *x* é aplicada.

As principais propriedades de uma máquina M são: *completamente especificada* se para cada estado de M existe uma transição para cada símbolo de entrada em X; *fortemente conectada* se para cada par de estados (S_i, S_j) existe uma seqüência de entrada que faz M transicionar de S_i para S_j ; *minimal* se o número de estados em M é menor ou igual ao número de estados para qualquer máquina M', que seja equivalente a M, isto é, que responda com uma saída idêntica para cada seqüência de entrada.

Dentre os métodos que têm sido propostos na literatura para geração de seqüências de teste para Máquinas de Estados Finitos, dois deles são utilizados no experimento apresentado neste capítulo: o método TT (Naito e Tsunoyama, 1981) e o método W (Chow, 1978).

O método TT (Transition Tour) objetiva, essencialmente, a construção de uma seqüência de teste que exercite toda transição S_i — x/y → S_j ∈ T. Este método, segundo Chow (1978), não é eficaz para revelar erros de transferência.

O Método W (Chow, 1978) apresenta um critério de teste denominado "Automata Theoretic", válido e confiável para estruturas de controle que possam ser modeladas com Máquinas de Estados Finitos, onde a próxima operação e o próximo estado dependem somente do estado corrente e da entrada, ou seja, não existem variáveis de controle ou contadores manipulados pelas operações que possam influenciar o seqüenciamento das operações. Assume-se ainda que as máquinas sejam: completamente especificadas, minimais, tenham um estado inicial fixo e que todo estado seja alcançável. Esse método consiste de três passos principais: 1) estimativa do número de estados do projeto correto, 2) geração de seqüências de teste baseada no projeto e 3) verificação das respostas às seqüências geradas em 2). Segundo Chow, as seqüências de teste geradas por esse método, se respeitadas todas as condições para sua aplicação, garantem revelar todos os erros de seqüenciamento.

Na seção seguinte apresentam-se os operadores de mutação para MEF, que foram definidos inspirando-se na classificação de erros proposta por Chow (1978).

4.3 Definição dos Operadores de Mutação para Máquinas de Estados Finitos

Segundo Chow (1978), um dos maiores problemas do teste é determinar um critério de seleção de casos de teste que seja válido e confiável. Considerando-se M uma máquina correta e M' uma máquina que se deseja avaliar, ambas minimais, com o mesmo alfabeto de entrada, Chow classifica os erros de seqüenciamento em 3 tipos: *erros de transferência* (quando M' não é equivalente a M mas pode se tornar equivalente a M, mudando-se apenas a função de próximo estado de M'); *erros de operação* (quando M' não é equivalente a M mas pode se tornar equivalente a M, mudando-se apenas a função de saída de M') e *erros de estados extras ou ausentes* (quando, a fim de tornar M' equivalente a M, o número de estados em M' deve ser reduzido ou aumentado; como M' e M são minimais, um número diferente de estados implica que M' e M não são equivalentes).

Baseando-se nessa classificação proposta por Chow (1978) definiram-se os operadores de mutação para MEFs, que estão apresentados a seguir, de acordo com a estrutura da MEF que sofre a alteração quando o operador é aplicado, podendo ser uma transição, uma saída ou um estado.

Seja M uma MEF tal que:

X : conjunto de entradas x	$\|X\| = m$
E : conjunto de estados, com S_0 estado inicial	$\|E\| = k$
Y : conjunto de saídas y	$\|Y\| = n$

T : função de transição, $X \times E \rightarrow E$

O : função de saída, $X \times E \rightarrow Y$

Sejam:

$TS = \{ t \mid t: S_i \text{---} x/y \rightarrow S_j$ onde $S_i, S_j \in E ; x \in X$ e $y \in Y \}$, $|TS| \leq m.k$;

$TS(S_i , S_j) = \{ t \in TS \mid t: S_i \text{---} x/y \rightarrow S_j$ onde $S_i, S_j \in E ; x \in X$ e $y \in Y \}$

denota o conjunto de transições para um determinado par de estados S_i, S_j

$X(S_i, S_j) = \{ x \mid \exists\ t : S_i \text{---} x/y \rightarrow S_j , x \in X$ e $y \in Y \}$

denota o conjunto de entradas que causam uma transição de S_i para S_j

$O(S_i, S_j) = \{ y \in Y \mid \exists\ t : S_i \text{---} x/ y \rightarrow S_j , x \in X(S_i, S_j) \}$

denota o conjunto de todas as saídas produzidas considerando-se todas as transições de S_i para S_j

$O(S_i, S_j)|_x = \{ y \in Y \mid \exists\ t : S_i \text{---} x/ y \rightarrow S_j$, para um específico $x \in X(S_i, S_j) \}$

denota o conjunto de saídas produzidas por uma determinada entrada que causa uma transição de S_i para S_j

Cada operador de mutação gera um conjunto de mutantes M_r ; cada M_r é uma MEF (X_r , E_r , Y_r , T_r , O_r) , como definido em seguida. Observe que M_0 é a MEF original (X , E , Y , T , O).

Obs: no caso de MEF considera-se que existe, no máximo, um arco representando as possíveis transições do estado S_i para o estado S_j e que existe uma transição para cada evento que provoca uma mudança do estado S_i para o estado S_j.

- **Erros de Transições**

Operador 1: ***alteração do estado inicial*** **(TraIniStaAlt)**

Este operador altera o estado inicial da MEF de forma que em cada mutante um dos outros estados passa a ser o estado de inicialização.

Mutantes M_r, $0 \leq r \leq (k-1)$, são gerados da seguinte maneira:

$X_r = X$; $Y_r = Y$; $T_r = T$; $O_r = O$; e E_r contém os mesmos estados que E mas,

$S_r \neq S_0$, $1 \leq r \leq (k-1)$, é o estado inicial no mutante M_r .

Operador 2: *arco faltando* (TraArcDel)

Este operador exclui um arco da MEF. Considera-se que o arco representa um caminho de um estado para outro; dessa forma, todos os eventos que provocam uma transição entre os estados que estão conectados por esse caminho, são excluídos.

Mutantes M_r , $0 \leq r \leq k^2$, são gerados da seguinte maneira:

Para cada par de estados (S_i , S_j) tal que $TS(S_i, S_j) \neq \phi$, é gerado um mutante tal que

$X_r \subseteq X$; $Y_r \subseteq Y$; $E_r = E$; $O_r \neq O$ e $T_r \neq T$

$TS_r = TS - TS(S_i, S_j)$

Operador 3: *evento faltando* (TraEveDel)

Este operador exclui um evento que provoca a transição entre dois estados. No caso desse evento ser o único a provocar essa transição, a alteração gerada por esse operador é a mesma daquela gerada pelo operador arco faltando. No caso da transição entre os estados ser provocada por mais de um evento, os demais eventos continuam existindo.

Mutantes M_r , $0 \leq r \leq (m \cdot k)$, são gerados da seguinte maneira:

Para cada par de estados (S_i, S_j) definem-se $|X(S_i, S_j)|$ mutantes tal que

$X_r \subseteq X$; $Y_r \subseteq Y$; $E_r = E$; $O_r \neq O$ e $T_r \neq T$

$X_r(S_i, S_j) = X(S_i, S_j) - \{x\}$, $x \in X(S_i, S_j)$;

Assim:

$TS_r(S_i, S_j) = TS(S_i, S_j) - \{t\}$ onde $t : S_i — x / y \rightarrow S_j$, $x \in X(S_i, S_j)$, $y \in Y$

Se $|X(S_i, S_j)| = 1$ este operador é equivalente ao operador arco faltando e portanto, nenhum mutante é gerado.

Operador 4: *evento extra* (TraEveIns)

Este operador inclui em cada arco da MEF cada um dos outros eventos existentes que não provoca a transição entre os estados relacionados ao arco considerado.

Mutantes M_r , $0 \leq r \leq (k . (m - 1))$ são gerados da seguinte maneira:

Para cada par de estados (S_i , S_j) , definem-se $(m - d_{ij})$, onde $d_{ij} = |\cup(X(S_i, S_z))|$, $z = 1,...,k$, mutantes tal que

$X_r = X$; $Y_r = Y$; $E_r = E$; $O_r \neq O$ e $T_r \neq T$

$X_r(S_i, S_j) = X(S_i, S_j) \cup \{x_p\}$, $x_p \in X$ e $x_p \notin \cup(X(S_i, S_z))$, $z = 1,...,k$

Assim:

$TS_r(S_i , S_j) = TS(S_i , S_j) \cup \{t_r\}$,

$t_r : S_i — x_p / \lambda \rightarrow S_j$, (λ é a saída vazia)

tal que para cada $x_p \in X$, $\neg\exists\, t \in TS$, $t : S_i — x_p / y \rightarrow S_z$, $z = 1,...,k$

Operador 5: *evento trocado* (TraEveAlt)

Este operador troca cada evento da MEF pelos demais eventos existentes, desde que estes já não provoquem a transição entre os mesmos estados que estão relacionados ao evento que está sendo trocado.

Mutantes M_r , $0 \leq r \leq (k . (m^2 / 4))$ são gerados da seguinte maneira:

Para cada par de estados (S_i , S_j) , definem-se $d_{ij} . (m - d_{ij})$,
onde $d_{ij} = | \cup(X(S_i, S_z)) |$, $z = 1,...,k$, mutantes tal que

$X_r \subseteq X$; $Y_r = Y$; $E_r = E$; $O_r \neq O$ e $T_r \neq T$

$X_r (S_i, S_j) = X (S_i, S_j) - \{x_q\} \cup \{x_p\}$, $x_q \in X(S_i , S_j)$ e
$x_p \in X$, $x_p \notin \cup(X(S_i , S_z))$, $z = 1,...,k$

Assim:
$TS_r (S_i , S_j) = TS (S_i , S_j) - \{ t \} \cup \{ t_r \}$, onde

$t : S_i — x_q / y \rightarrow S_j$ e

$t_r : S_i — x_p / y \rightarrow S_j$

Operador 6: *destino trocado* (TraDesStaAlt)

Este operador troca o estado destino associado a cada um dos eventos pelos demais estados existentes na MEF.

Mutantes M_r , $0 \leq r \leq ((m . k) . (k - 1))$, são gerados da seguinte maneira:

Para cada par de estados (S_i, S_j) tal que $TS (S_i, S_j) \neq \phi$, definem-se $(k-1)$ mutantes tal que

$X_r = X$; $Y_r = Y$; $E_r = E$; $O_r = O$ e $T_r \neq T$

$TS_r (S_i, S_j) = TS (S_i, S_j) - \{ t \}$, onde $t : S_i — x / y \rightarrow S_j$ e

$TS_r (S_i, S_k) = TS (S_i, S_k) \cup \{ t_r \}$, onde

$t_r : S_i — x / y \rightarrow S_k$, para todo $S_k \in E$, $S_k \neq S_j$

S_j é trocado por todos os outros estados de E.

- **Erros de Saída**

Operador 7: *saída faltando* (OutDel)

Este operador remove o símbolo de saída associado a cada evento. No caso do evento possuir vários símbolos de saída associados a ele (saída composta), é retirado um símbolo de cada vez.

Mutantes M_r , $0 \leq r \leq ((m . k) . n)$, são gerados da seguinte maneira:

Para cada par de estados (S_i, S_j) e para cada $x \in X(S_i, S_j)$, definem-se $| O(S_i, S_j)|_x |$ mutantes tal que

$X_r = X$; $Y_r \subseteq Y$; $E_r = E$; $T_r = T$ e $O_r \neq O$

$(O(S_i, S_j)|_x)_r = O(S_i, S_j)|_x - \{ y_p \}$, $y_p \in O(S_i, S_j)|_x$

Assim:

$TS_r (S_i, S_j) = TS (S_i, S_j) - \{ t \} \cup \{ t_r \}$,

$t : S_i — x / (O(S_i, S_j)|_x) \rightarrow S_j$ e $t_r : S_i — x / (O(S_i, S_j)|_x)_r \rightarrow S_j$

Operador 8: *saída trocada* (OutAlt)

Este operador altera a saída gerada pelos eventos, pelas demais saídas existentes. No caso de saídas compostas por vários símbolos, a troca é realizada em cada um deles, desde que a saída que esteja sendo colocada não faça parte das demais saídas já associadas ao mesmo evento.

Mutantes M_r , $0 \leq r \leq ((m \,.\, k) \,.\, (n - 1))$, são gerados da seguinte maneira:

Para cada par de estados (S_i, S_j) e para cada $x \in X(S_i, S_j)$, definem-se $| O(S_i, S_j)|_x | \,.\, (n - | O(S_i, S_j)|_x |)$ mutantes tal que

$X_r = X$; $Y_r \subseteq Y$; $E_r = E$; $T_r = T$ e $O_r \neq O$

$(O(S_i, S_j)|_x)_r = O(S_i, S_j)|_x - \{ y_q \} \cup \{ y_p \}$,

$y_q \in O(S_i, S_j)|_x$ e $y_p \in Y$, $y_p \notin O(S_i, S_j)|_x$

Assim:

$TS_r (S_i, S_j) = TS (S_i, S_j) - \{ t \} \cup \{ t_r \}$, onde

$t : S_i — x / (O(S_i, S_j)|_x) \rightarrow S_j$ e $t_r : S_i — x / (O(S_i, S_j)|_x)_r \rightarrow S_j$

- **Erros de Estados**

Operador 9: *estado faltando* (StaDel)

Este operador faz uma junção de dois estados em um único, desde que exista um arco conectando os mesmos. O estado resultante dessa junção passa a ser a origem e/ou destino de todos os arcos que tinham como origem e/ou destino os estados que foram agrupados.

Mutantes M_r, $0 \leq r \leq C_2^k$, são gerados da seguinte maneira:

Para cada par de estados (S_i, S_j) tal que $\exists\, t : S_i \text{ — } x / y \rightarrow S_j$ define-se um mutante tal que

$X_r = X$; $Y_r = Y$; $E_r \neq E$; $O_r \neq O$ e $T_r \neq T$

$E_r = E - \{ S_i , S_j \} \cup \{ S' \}$, onde S' é a junção de S_i com S_j

$X_r(S_z, S') = X(S_z, S_i) \cup X(S_z, S_j)$

$X_r(S', S_z) = X(S_i, S_z) \cup X(S_j, S_z)$

$X_r(S', S') = X(S_i, S_i) \cup X(S_j, S_j)$

Assim:

$TS_r(S_z, S') = TS(S_z, S_i) \cup TS(S_z, S_j)$

$TS_r(S', S_z) = TS(S_i, S_z) \cup TS(S_j, S_z)$

$TS_r(S', S') = TS(S_i, S_i) \cup TS(S_j, S_j)$

para todo z tal que: $TS(S_i , S_z) \neq \phi$

$TS(S_j , S_z) \neq \phi$

$TS(S_z , S_i) \neq \phi$

$TS(S_z , S_j) \neq \phi$

Operador 10: *estado extra*

Este operador define um novo estado na MEF, tornando-a não minimal. Neste trabalho foram definidas quatro abordagens para criar o estado extra, especificadas a seguir:

a) estado extra: *isola saída relevante* (StaInsOutSep)

Este operador desvia para um estado extra todos os eventos da MEF, que emitem uma determinada saída que passa a ser tratada separadamente, como por exemplo, mensagem de erro.

Mutantes M_r, $0 \leq r \leq n$, são gerados da seguinte maneira:

Para cada $y_p \in Y$ gera-se um mutante tal que

$X_r = X$; $Y_r = Y$; $E_r \neq E$; $O_r \neq O$ e $T_r \neq T$

$E_r = E \cup \{ S' \}$

$TS_r (S_i, S_j) = TS (S_i, S_j) - \{ t \mid t \in TS(S_i, S_j)$ e $t : S_i \text{---} x / y_p \rightarrow S_j \}$, $\forall S_i, S_j \in E$

$TS_r (S_i, S') = \{ t_r \mid t_r : S_i \text{---} x / y_p \rightarrow S'$, tal que $\exists x \in X$ e $t \in TS$, $t : S_i \text{---} x / y_p \rightarrow S_j$, $S_i, S_j \in E \}$

$TS_r (S', S_d) = \{ t_r \mid t_r : S' \text{---} x / y \rightarrow S_d$ tal que $\exists t \in TS$, $t : S_j \text{---} x / y \rightarrow S_d$, $j \neq d$, $S_j, S_d \in E \}$

$TS_r (S', S') = \{ t_r \mid t_r : S' \text{---} x / y \rightarrow S'$ e $\exists t : S_j \text{---} x / y \rightarrow S_j$, $S_j \in E \}$

b) estado extra: *desmembra transição com mais de um evento* (StaInsTraSep)

Este operador cria um estado extra e direciona para esse novo estado um dos eventos associados a uma transição que possui mais de um evento, considerando todas as transições que satisfazem essa condição na MEF.

Mutantes M_r, $0 \leq r \leq (m \cdot k)$ são gerados da seguinte maneira:

Para cada par de estados (S_i, S_j) onde $| TS(S_i, S_j) | > 1$, definem-se $| TS(S_i, S_j) |$ mutantes tal que

$X_r = X$; $Y_r = Y$; $E_r \neq E$; $O_r \neq O$ e $T_r \neq T$

$E_r = E \cup \{ S' \}$

$TS_r (S_i, S_j) = TS (S_i, S_j) - \{ t \}$, $t \in TS(S_i, S_j)$, $t : S_i \text{—} x_q / y \rightarrow S_j$, $S_i , S_j \in E$

$| TS_r (S_i, S') | = 1$ e
$TS_r (S_i, S') = \{ t_r \}$, $t_r : S_i \text{—} x_q / y \rightarrow S'$

$TS_r (S', S_d) = \{ t_r | t_r : S' \text{—} x / y \rightarrow S_d$ tal que $\exists$ $t \in TS$, $t : S_j \text{—} x / y \rightarrow S_d$, $j \neq d$, $S_j , S_d \in E \}$

$TS_r (S', S') = \{ t_r | t_r : S' \text{—} x / y \rightarrow S'$ e $\exists$ $t : S_j \text{—} x / y \rightarrow S_j$, $S_j \in E \}$

c) estado extra: *separa eventos com mesma saída de uma transição*
(StaInsTraOutSep)

Este operador cria um estado extra e direciona para esse estado os eventos de uma mesma transição que produzem a mesma saída, considerando todas as transições que possuem vários eventos associados a ela e que estes gerem, pelo menos, duas saídas diferentes.

Mutantes M_r, $0 \leq r \leq (k . n)$ são gerados da seguinte maneira:

Para cada par de estados (S_i, S_j) onde $| X (S_i, S_j) | > | O (S_i, S_j) |$, e $| O (S_i, S_j) | > 2$, definem-se $| O (S_i, S_j) |$ mutantes tal que

$X_r = X$; $Y_r = Y$; $E_r \neq E$; $O_r \neq O$ e $T_r \neq T$

$E_r = E \cup \{ S' \}$

$TS_r(S_i, S_j) = TS(S_i, S_j) - \{ t \mid t : S_i — x / y_q \rightarrow S_j \ , \forall x \in X(S_i, S_j) , \ S_i, S_j \in E \}$

$TS_r(S_i, S') = \{ t_r \mid t_r : S_i — x / y_q \rightarrow S' \ , \ \text{tal que} \ \exists \ t \in TS , \ t : S_i — x / y_q \rightarrow S_j \ ,$
$y_q \in O(S_i, S_j) , \ S_i, S_j \in E \}$

$TS_r(S', S_d) = \{ t_r \mid t_r : S' — x / y \rightarrow S_d \ \text{tal que} \ \exists \ t : S_j — x / y \rightarrow S_d , \ j \neq d ,$
$S_j, S_d \in E \}$

$TS_r(S', S') = \{ t_r \mid t_r : S' — x / y \rightarrow S' \ \text{e} \ \exists \ t : S_j — x / y \rightarrow S_j , \ S_j \in E \}$

d) estado extra: *combina transições* (StaInsTraComb)

Este operador considera um terno de estados que estão conectados através de transições e cria um estado extra, direcionando para ele as várias possibilidades de combinação entre as transições que conectam os três estados considerados.

Mutantes M_r , $0 \leq r \leq (C_3^k . m . 2^m)$, são gerados da seguinte maneira:

Para cada terno de estados (S_i, S_j, S_d) tal que $| TS(S_i, S_j) | \geq 1$ e $| TS(S_j, S_d) | \geq 1$,
definem-se $| TS(S_i, S_j) | . | Pot (TS(S_j, S_d)) |$ mutantes,
onde $Pot (TS(S_j, S_d)$ é o conjunto potência de $(TS(S_j, S_d))$.
Ou seja, definem-se $| TS(S_i, S_j) | . 2^{| TS(S_j, S_d) |}$ mutantes tal que em cada mutante associado ao terno de estados tem-se

$X_r = X ; Y_r = Y ; E_r \neq E ; O_r \neq O$ e $T_r \neq T$

$E_r = E \cup \{ S' \}$

$TS_r(S_i, S_j) = TS(S_i, S_j) - \{ t \}$, $t \in TS(S_i, S_j)$, $t : S_i — x_q / y \rightarrow S_j$

$| TS_r(S_i, S') | = 1$

$TS_r(S_i, S') = \{ t_r \}$, $t_r : S_i — x_q / y \rightarrow S'$

$TS_r(S', S_j) = \phi$

$TS_r(S', S_d) = t\text{-pot}$, $t\text{-pot} \in Pot (TS(S_j, S_d))$

$TS_r(S_j, S_d) = TS(S_j, S_d) - \{ t\text{-pot} \}$

$TS_r(S', S') = \phi$

Observa-se que para cada $t \in TS(S_i, S_j)$ geram-se tantos mutantes quanto for a cardinalidade do conjunto $Pot (TS(S_j, S_d))$.

Os operadores foram apresentados segundo um agrupamento que corresponde à estrutura alterada por eles, ou seja, transição, estado ou saída. No entanto, como já foi mencionado, a definição desses operadores foi inspirada na classificação de erros proposta por Chow (1978). Assim, na Tabela 4.1, adiante, apresentam-se os operadores de mutação relacionando-os com a classificação de Chow (1978).

Na seção seguinte explora-se a adequação de seqüências de teste geradas pelo método W e pelo método TT à Análise de Mutantes, considerando-se para a aplicação do critério, o conjunto de operadores de mutação definido anteriormente. Observa-se que a definição do operador *estado extra* foi feita com o objetivo de explorar algumas situações de erro que ocorrem ao serem gerados mutantes de ordem 2, como será exemplificado.

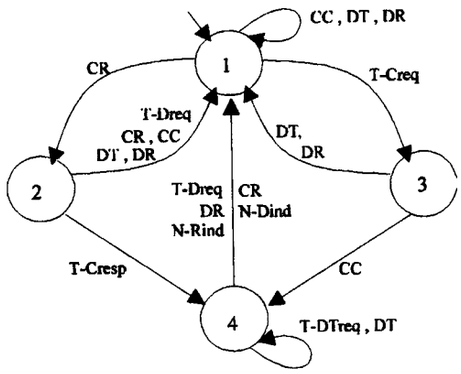
Tabela 4.1 - Agrupamento dos Operadores Segundo a Nomenclatura Utilizada por Chow (1978)

agrupamento apresentado	*Classificação Chow* / *Operadores*	erros de transferência	erros de operação	erros de estados extras/ausentes
erros de transição	1.alteração do estado inicial	x		
	2.arco faltando	x		
	3.evento faltando	x		
	4.evento extra	x		
	5.evento trocado	x		
	6.destino trocado	x		
erros de saída	7.saída faltando		x	
	8.saída trocada		x	
erros de estados	9.estado faltando			x
	10.estado extra			x

4.4 Um Exemplo de Aplicação do Critério Análise de Mutantes em Máquinas de Estados Finitos

Com base no conjunto de operadores especificados na seção anterior, a aplicação da Análise de Mutantes em Máquinas de Estados Finitos foi avaliada tomando-se como exemplo o Diagrama de Estado do Protocolo de Transporte Classe 0 da ISO (Gabos e Stiubiener, 1990), apresentado na Figura 4.1, e as seqüências de teste geradas pelo método W (Chow, 1978) e pelo método TT (Naito e Tsunoyama, 1981), apresentadas também em (Gabos e Stiubiener, 1990).

Na Figura 4.2 apresentam-se exemplos de mutantes de ordem 1 para alguns desses operadores e na Tabela 4.2 apresenta-se uma síntese da aplicação da Análise de Mutantes ao exemplo ilustrado na Figura 4.1.



Estado / Entrada	1 livre	2 espera conf. do usuário	3 espera conf. da rede	4 trans. de dados
T_Creq	3/CR	-	-	-
T_Dreq	-	1/DR	-	1/N_Dreq
T_Cresp	-	4/CC	-	-
T_DTreq	-	-	-	4/DT
CR	2/T_Cind	1/ERR	-	1/ERR
CC	1/-	1/ERR	4/T_Cconf	-
DT	1/-	1/ERR	1/-	4/T_DTind
DR	1/-	1/ERR	1/T_Dind, N_Dreq	1/N_Dreq
N_Dind	-	-	-	1/T_Dind
N_Rind	-	-	-	1/T_Dind

Seqüência TT = CR, T_Dreq, CC, DT, DR, CR, CR, T_Creq, DT, CR, T_Cresp, T_DTreq, DT, T_Dreq, CR, CC, CR, DT, CR, DR, T_Creq, DR, T_Creq, CC, CR, CR, T_Cresp, DR, CR, T_Cresp, N_Dind, CR, T_Cresp, N_Rind

Seqüência W = (DR, T_Creq.DR, CR.DR, CC.DR, DT.DR, DR.DR,
CR.(T_Dreq, T_Cresp, CR, CC, DT, DR).DR,
T_Creq.(CC, DT, DR).DR,
CR.T_Cresp.(T_Dreq, T_DTreq, CR, DT, DR, N_Dind, N_Rind).DR)

Figura 4.1 - Máquina de Estado, Tabela de Estado, Seqüências TT e W

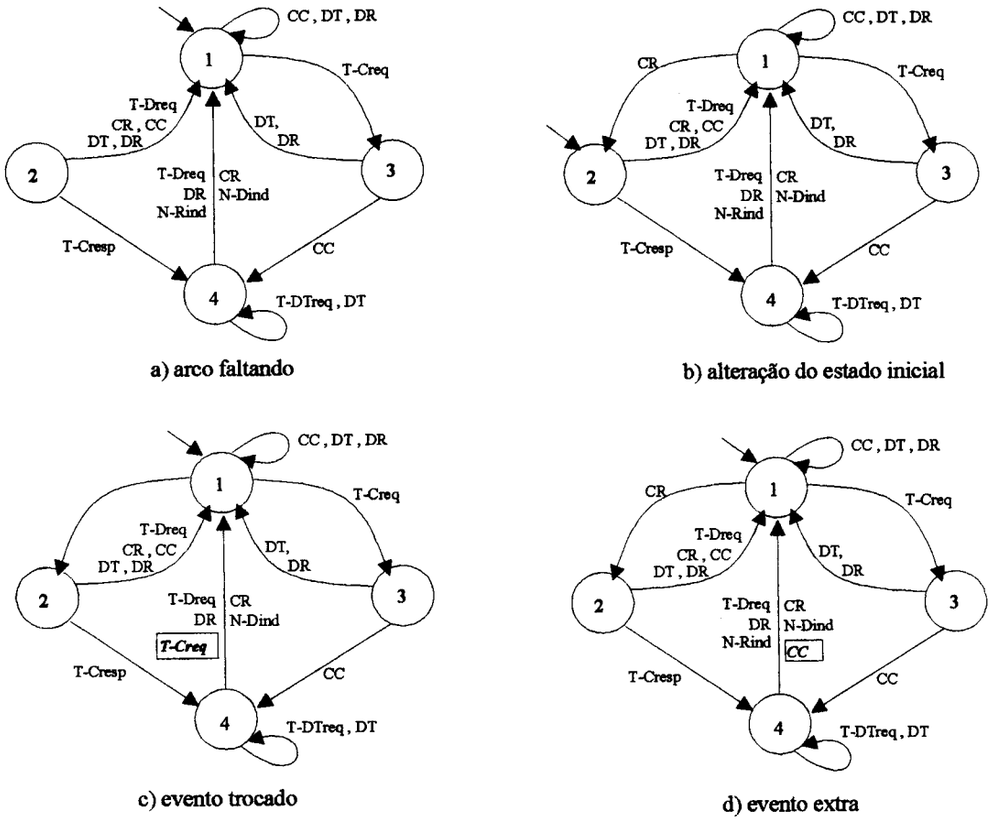


Figura 4.2- Exemplos de Mutantes Gerados por Alguns Operadores

Primeiramente, nota-se que, na prática, as especificações não satisfazem as restrições iniciais para a aplicação dos métodos de geração de seqüências de teste que, em geral, são: máquina completamente especificada, minimal, fortemente conectada e determinística. Por exemplo, a Máquina de Estados Finitos considerada não satisfaz a condição de ser completamente especificada.

Caso todas as restrições iniciais fossem satisfeitas, as seqüências geradas pelo método W, em teoria, deveriam ter um escore de mutação de 100% se considerássemos somente 1-mutante pois, se as Máquinas de Estados Finitos respeitassem as hipóteses para a aplicação do método W, todos os erros de seqüenciamento no projeto seriam revelados (Chow, 1978) e, conseqüentemente, todos os mutantes deveriam ser distinguidos da Máquina M original, por refletirem as classes de erros consideradas: erro de operação, erro de transferência e erro

de estados extras ou ausentes. No entanto, como é apresentado a seguir, na Tabela 4.2, os escores obtidos não foram de 100% em decorrência, justamente, da MEF não satisfazer todos os requisitos para a aplicação do método.

Tabela 4.2 - Resultados da Aplicação da Análise de Mutantes ao Exemplo da Figura 4.1

			método W		método TT	
operador	total	equiv.	vivos	mortos	vivos	mortos
1. alteração-do-estado-inicial	3	0	0	3	0	3
2. arco-faltando	9	1	0	8	0	8
3. evento-faltando	21	3	0	18	0	18
4. evento-extra	43	5	38	0	38	0
5. evento-trocado	91	15	0	76	0	76
6. destino-trocado	63	0	0	63	5	58
7. saída-trocada	202	0	0	202	0	202
8. saída-faltando	18	0	0	18	0	18
9. estado-faltando	1	0	0	1	0	1
Total	451	24	38	389	43	384
Escore			0.9110		0.8992	

Essa Tabela contém uma síntese dos resultados obtidos. Observa-se que para a MEF em questão, quando aplicados os operadores de mutação definidos, caso não fosse considerado o aspecto de não determinismo, teriam sido gerados 78 mutantes não determinísticos: 26 para o operador 4, 48 para o operador 5 e 4 para o operador 9. No entanto, se considerado esse aspecto, como está sendo feito neste trabalho, esses mutantes simplesmente não são gerados, restringindo-se a aplicação do critério às máquinas determinísticas, uma vez que é pouco provável que o comportamento das não determinísticas possa ser reproduzido.

Para a distinção entre o comportamento da Máquina original M e dos mutantes, foi utilizada a saída gerada pelos eventos. Em relação ao exemplo, mesmo não sendo respeitadas as condições iniciais, a seqüência gerada pelo método W obteve um escore de mutação bastante alto, considerando-se os valores apresentados na Tabela 4.2.

Observa-se que as seqüências geradas pelo método W e pelo método TT tiveram praticamente a mesma adequação em relação à Análise de Mutantes; isto é decorrente do fato da máquina ser simples e simétrica. Sem dúvida, este não seria o comportamento em geral, pois as seqüências geradas pelo método TT não são tão eficazes na revelação de certas classes de erros quanto o método W (Chow, 1978). A seguir, discute-se mais detalhadamente a adequação da seqüência gerada pelo método W, em relação à Análise de Mutantes.

Os mutantes que permanecem vivos, num total de 38, são aqueles decorrentes da aplicação do operador *evento extra*. Isso acontece pelo fato da Máquina de Estados Finitos não ser completamente especificada. Também em decorrência desse fato, observou-se a existência de muitos mutantes equivalentes, num total de 15, pela aplicação do operador *evento trocado*, que explora a mesma característica.

Para uma classe de Máquinas de Estados Finitos, as MEFs completamente especificadas, o problema de equivalência é decidível (Kohavi, 1978; Nakazato et al, 1996). No entanto, na prática, as MEFs não satisfazem essa propriedade, não possibilitando, conseqüentemente, a determinação automatizada da equivalência. No contexto de Máquinas de Estados Finitos Estendidas, em geral, esse aspecto é indecidível e o estabelecimento de heurísticas ou de mecanismos que facilitem o testador na execução dessa tarefa é um ponto relevante para a aplicação da Análise de Mutantes.

O fato de que, para esse exemplo, obteve-se um escore de mutação bastante alto, mesmo não satisfazendo as condições iniciais para a aplicação do método W, não implica, seguramente, que esta seja a situação geral. Normalmente, lida-se com máquinas mais complexas, não minimais, com variáveis de estado, aspectos que, sem dúvida, tornam mais fraca a seqüência de teste

gerada pelo método W, do ponto de vista de validação, o que levaria, certamente, a um escore inferior a 100%. Observe-se que mutantes seriam gerados com base nessas variáveis de estado o que, em nível de programa, está associado aos operadores que mais geram mutantes.

Para Máquinas de Estados Finitos, assim como para programas, também é possível gerar mutantes com a aplicação de mais de um operador de mutação de uma só vez, obtendo-se mutantes de ordem k (k-mutante). Um ponto importante que deve ser ressaltado é em relação a k-mutantes e a propriedade de máquina minimal exigida pelo método W. No exemplo apresentado existem evidências de que, em geral, o efeito de acoplamento seria válido também para a Análise de Mutantes baseada em Máquinas de Estados Finitos, se bem que essa hipótese deva ser avaliada para um número significativo de modelos. No entanto, a combinação do operador *estado extra*, levando a máquinas mutantes equivalentes não minimais, com os demais operadores de mutação, gera mutantes para os quais as seqüências de teste são ineficazes, ou seja, não são capazes de distinguir entre o comportamento de M e desses mutantes.

Considere uma máquina M_e equivalente à máquina M, da Figura 4.1, gerada pelo operador *estado extra* - tipo (a), ilustrada na Figura 4.3a. Pode-se considerar que essa máquina tenha sido concebida pressupostamente de forma a facilitar o entendimento e a manutenção da máquina M, no que concerne à emissão de mensagens de erros. A máquina M_e é uma máquina não minimal, pois os estados *1* e *Err* são equivalentes. Se considerarmos agora mutação na máquina M_e, ilustrada na Figura 4.3b, as seqüências geradas pelo método W, com base na máquina M, não distinguiriam uma boa parte desses mutantes: considerando apenas essa ocorrência do operador *arco faltando*, denotada nesse exemplo específico, de 5 mutantes apenas 2 são eliminados dando, aproximadamente, um escore de mutação de 40%.

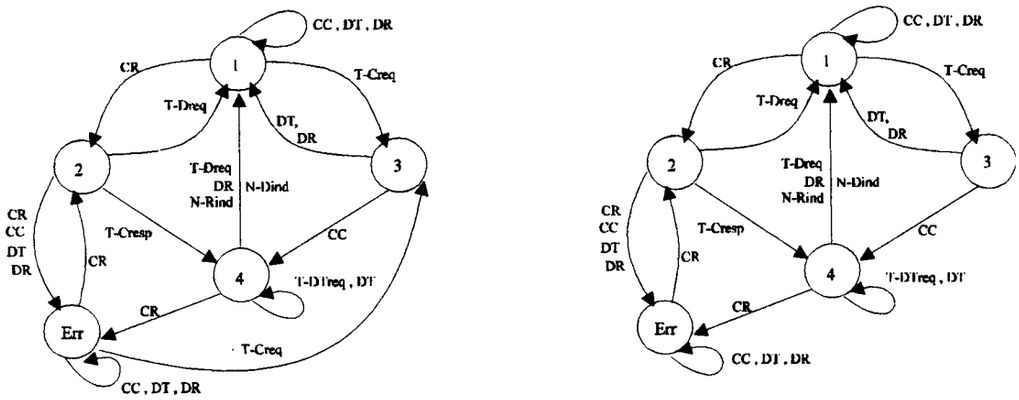


Figura 4.3 - (a) - Aplicação do Operador Estado Extra para Isolar Situação de Erro
(b) - Mutante de Ordem 2: Aplicação do Operador Arco Faltando sobre o Operador Estado Extra

Considere novamente o operador de mutação *estado extra*, e a mutação do tipo (b). A aplicação desse operador sobre a MEF da Figura 4.1 resulta num mutante equivalente, conforme ilustrado na Figura 4.4. A mutação foi aplicada na transição que vai do estado 3 para o estado 1, separando-se os eventos associados a ela, direcionando o evento DT para o estado 1', que corresponde ao estado extra, equivalente ao estado 1.

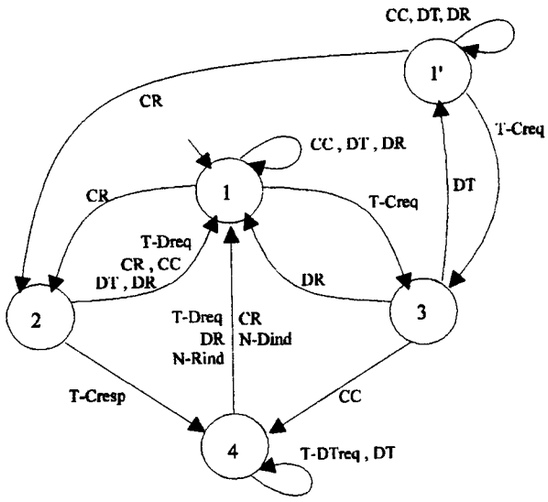


Figura 4.4 - Aplicação do Operador Estado Extra para Desmembrar Transições com mais de um Evento

Novamente aplicando-se o operador *estado extra*, do tipo (c), sobre a MEF da Figura 4.1, obtém-se o mutante equivalente, ilustrado na Figura 4.5. A mutação foi aplicada na transição que vai do estado 4 para o estado 1, direcionando para o estado extra, 1', os eventos T-Dreq e DR que produzem a mesma saída.

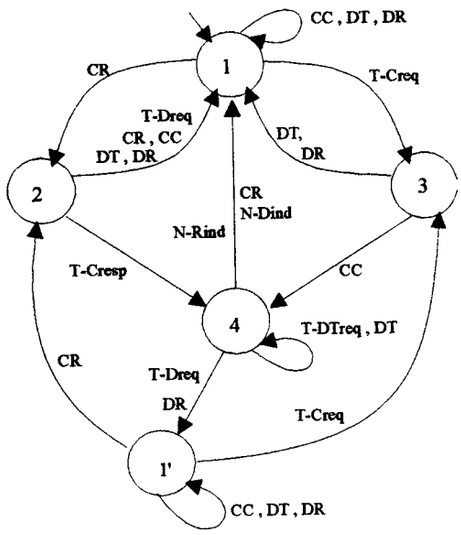
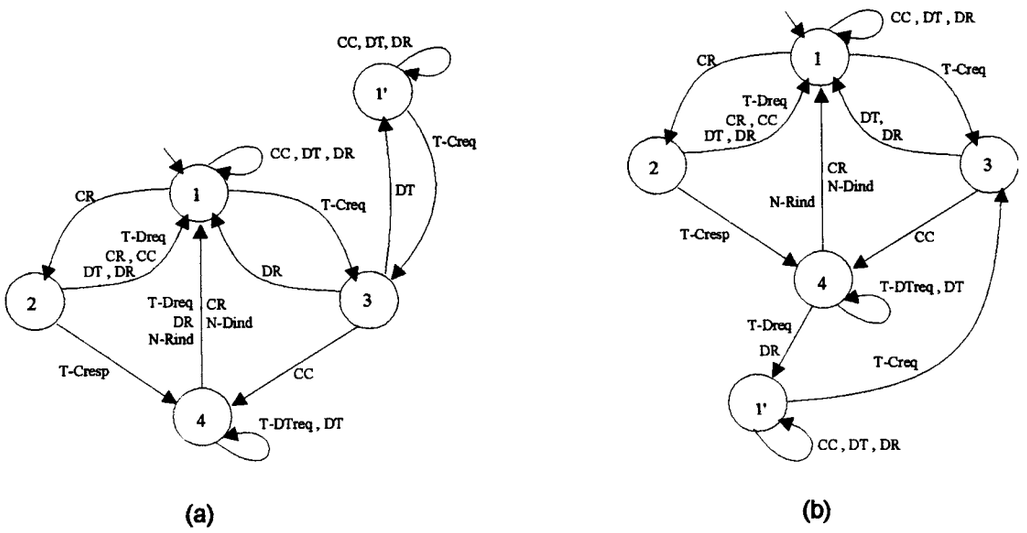


Figura 4.5 - Aplicação do Operador Estado Extra para Separar Eventos de Uma Mesma Transição que Produzem a Mesma Saída

Após a geração desses mutantes equivalentes não minimais, seriam aplicados os demais operadores de mutação gerando-se, dessa forma, mutantes de ordem 2 (2-mutantes). Considere um exemplo de 2-mutante, ilustrado na Figura 4.6b, gerado pela aplicação do operador *arco faltando* sobre esses mutantes equivalentes, em particular sobre o mutante da Figura 4.5, considerando-se apenas o conjunto de transições que incluem o estado extra. O escore de mutação nesse caso é, de aproximadamente, 25%, um escore relativamente baixo. Por exemplo, o comportamento do 2-mutante da Figura 4.6a, gerado a partir do mutante da Figura 4.4, não é distinguido pela seqüência de teste gerada pelo método W, do comportamento da máquina original.



Figuras 4.6 - Aplicação do Operador Arco Faltando sobre o Operador Estado Extra (a) - em relação à Figura 4.4 e (b) - em relação à Figura 4.5

Com esse exemplo, fornecem-se evidências de que a Análise de Mutantes no contexto de teste baseado em Máquinas de Estados Finitos consiste em um mecanismo complementar aos métodos de geração de seqüências de teste, uma vez que, em geral, as máquinas de estado não satisfazem as restrições exigidas para a aplicação desses métodos, tornando a atividade de validação menos confiável.

4.5 Aplicação da Análise de Mutantes Alternativa em Máquinas de Estados Finitos

Nesta seção apresenta-se uma síntese dos resultados da aplicação dos critérios alternativos da Análise de Mutantes ao exemplo considerado na seção anterior. A Mutação Aleatória (n%) considerada foi a mutação 10% e a Mutação Restrita restringiu o conjunto de operadores de mutação aos operadores *(arco/evento/estado)-faltando* e *saída trocada*. Os conjuntos de teste associados à mutação restrita estão referenciados com o sufixo "4op" por terem sido selecionados 4 tipos de operadores.

Foram utilizados conjuntos de casos de teste adequados para cada um dos critérios considerados, isto é, para a Mutação Aleatória e para a Mutação Restrita. Inicialmente, analisou-se a adequação das seqüências W e TT em relação aos mutantes gerados pelos critérios alternativos e esses conjuntos de teste foram completados a fim de obterem-se conjuntos de teste adequados para cada um dos critérios alternativos.

No caso da seqüência gerada pelo método W, utilizaram-se para cada um dos critérios propostos, dois conjuntos de teste: um deles minimizado (W10%min e W4opmin), onde constam apenas as seqüências do método W que efetivamente matam todos os mutantes gerados pelos critérios alternativos e o outro, que utiliza toda a seqüência W (W10% e W4op), uma vez que ela já não é suficiente para atingir um escore de 100%. Considerando-se a mutação 10%, nos dois casos foi necessário o acréscimo de um conjunto de seqüências para matar todos os mutantes que permaneceram vivos quando da aplicação da seqüência W aos mutantes desse critério alternativo. No caso da mutação restrita, a seqüência W original é adequada ao critério, como pode ser visto na Tabela 4.3, sendo inclusive menor, quando considerado o conjunto minimizado.

No caso da seqüência gerada pelo método TT, quando aplicado o critério de mutação 10%, a seqüência original também foi acrescida de um conjunto de eventos para tornar-se adequada a esse critério. Já no caso da mutação restrita, a seqüência original é adequada ao critério.

Para o cálculo da métrica C_1, as seqüências W e TT originais foram acrescidas de outras seqüências e eventos, respectivamente, para torná-las adequadas à mutação ("full mutation"), uma vez que essa métrica analisa o tamanho das seqüências adequadas aos critérios alternativos em relação ao tamanho da seqüência adequada à Análise de Mutantes. Assim, a seqüência W adequada à mutação possui 41 sub-seqüências e a seqüência TT adequada à mutação possui 94 eventos.

A seguir, analisa-se a adequação de cada um dos conjuntos de teste adequados aos critérios alternativos em relação ao critério Análise de Mutantes. Os resultados obtidos estão sumarizados na Tabela 4.3.

Tabela 4.3 - Dados de Análise do Fator "Strength"

critério (*nº de mutantes*)		mutação 10%	(arco/evento/estado)-faltando saída-trocada	mutação
seqüência	*compr.*	(46)	(233)	(451)
W	22	0.9110	1	0.9110
TT	34	0.8889	1	0.8992
W10%min	25	1	0.9563	0.8829
W10%	26	1	1	0.9227
TT10%	42	1	0.9083	0.8384
W4opmin	20	0.8889	1	0.9039
W4op	22	0.9110	1	0.9110
TT4op	34	0.8889	1	0.8992

A partir da Tabela 4.3 pode-se computar as métricas C_1, C_2 e C_3, apresentadas na Seção 3.3.1, para cada um dos critérios de mutação alternativa definidos, em relação às seqüências W e TT, que são apresentadas nas Tabelas 4.4 e 4.5, respectivamente.

Tabela 4.4 - Dados de Custo e "Strength" em Relação à Seqüência W

Métrica \ *Mutação*		mutação 10%	(arco/evento/estado)-faltando saída-trocada
Custo	C_1	36.59%	46.34%
	C_2	89.80%	48.34%
Strength	C_3	92.27%	91.10%

Tabela 4.5 - Dados de Custo e "Strength" em Relação à Seqüência TT

Métrica \ *Mutação*		mutação 10%	(arco/evento/estado)-faltando saída-trocada
Custo	C_1	55.32%	63.83%
	C_2	89.80%	48.34%
Strength	C_3	83.84%	89.92%

Pela Tabela 4.4 observa-se que utilizando-se os critérios de mutação alternativa, a seqüência adequada a esses critérios é uma subseqüência da seqüência W e, além disso, o número de mutantes analisados é menor. Por exemplo, em termos de tamanho, a seqüência adequada à mutação 10% é reduzida em 36.59%, em relação à seqüência W adequada ao critério Análise de Mutantes e a seqüência adequada à mutação restrita é reduzida em 46.34%. Em termos do número de mutantes analisados, a mutação 10% provoca uma redução de 89.80%, enquanto que a mutação restrita provoca uma redução de 48.34%.

Ainda pela Tabela 4.4, em relação ao fator "strength", tomando por base a Análise de Mutantes, pode-se observar que conjuntos de teste adequados aos critérios "alternativos" apresentam um nível de adequação satisfatório em relação ao critério Análise de Mutantes "tradicional". A mutação 10% apresentou um escore de mutação maior que o da mutação restrita *(arco/evento/estado)-faltando*

e saída trocada. Isso se deve ao fato de que ao selecionar-se 10% de cada tipo de mutante, foram selecionados alguns mutantes do operador *evento extra*, onde vários deles permaneceram vivos, e sendo necessário tornar a seqüência W adequada à mutação 10%, novas sub-seqüências foram adicionadas à seqüência original, tornando-a mais eficaz na detecção de erros. O mesmo não ocorre na mutação restrita pois, dos operadores selecionados, a seqüência W original é capaz de distinguir todos os mutantes, fazendo com que o escore associado a esse critério seja o mesmo do critério Análise de Mutantes, de 91.10%. Isso vem reafirmar o fato de que a escolha e definição dos operadores de mutação é o ponto chave do critério Análise de Mutantes.

A mesma análise deve ser conduzida para a interpretação dos dados da Tabela 4.5, onde é avaliada a seqüência gerada pelo método TT. Resumidamente, os resultados apresentados mostram que obteve-se uma redução de custo nos dois critérios alternativos, tanto no que diz respeito ao tamanho da seqüência de teste como também ao número de mutantes examinados. Neste caso, a mutação 10% apresentou um menor escore de mutação do que a mutação restrita *(arco/evento/estado)-faltando* e *saída trocada*. Isto indica que para uma comparação efetiva das mutações alternativas no contexto de Máquinas de Estados Finitos e, em geral, no contexto de Sistemas Reativos, um estudo empírico mais abrangente deve ser planejado e conduzido.

4.6 Considerações Finais

Apresentou-se neste capítulo a aplicação do critério Análise de Mutantes na validação de especificações baseadas em Máquinas de Estados Finitos. Para tanto, definiu-se um conjunto de operadores de mutação com base na classificação de erros para MEFs, proposta por Chow (1978), que identifica três classes: erros de transferência, erros de operação e erros de estados extras/ausentes.

O critério foi aplicado a uma MEF que especifica um protocolo de transporte e que, como a maioria das especificações encontradas na prática, não

satisfaz os requisitos iniciais para a utilização dos métodos tradicionais de geração de seqüências de teste para MEFs. Com base no conjunto de operadores definidos, foram gerados os mutantes dessa MEF e foram aplicadas a eles duas seqüências de teste: uma gerada pelo método W e outra gerada pelo método TT. O que se pode observar foi que nem todos os mutantes foram mortos com essas seqüências de teste, ou seja, elas não se mostraram 100% adequadas à Análise de Mutantes, justamente pelo fato dos requisitos iniciais não serem satisfeitos pela MEF. Esse fato leva a evidências de que o critério Análise de Mutantes pode ser considerado uma forma complementar de teste nesse contexto.

Com a finalidade de avaliar o uso da mutação alternativa na validação de especificações baseadas em MEFs, ao mesmo exemplo foram aplicados os critérios de Mutação Aleatória e Mutação Restrita. Os resultados desse experimento foram semelhantes àqueles obtidos quando da utilização da Análise de Mutantes Alternativa no teste de programas. Ou seja, obteve-se uma redução em termos de custo, tanto no que diz respeito ao tamanho das seqüências de teste, como também ao número de mutantes analisados, sem se perder muito na eficácia do critério. Esse resultado incentiva ainda mais a continuidade desta pesquisa, mostrando que o critério Análise de Mutantes pode ser aplicado de forma incremental, e direcionado para a identificação de determinados tipos de erros que se desejam detectar na especificação.

Um aspecto que merece uma maior investigação é o efeito de acoplamento. Como pode ser observado pelos exemplos apresentados neste capítulo, ao serem gerados mutantes de ordem 2, onde a aplicação do primeiro operador produz uma MEF equivalente à original e o segundo operador produz uma MEF errônea, as seqüências de teste utilizadas tornaram-se ainda menos eficazes. Assim, todos os mutantes de ordem 2 devem ser gerados não só para esse exemplo, como também para outras especificações, de forma a se poder chegar em resultados mais conclusivos.

CAPÍTULO 5

ANÁLISE DE MUTANTES APLICADA EM REDES DE PETRI

5.1 Considerações Iniciais

Rede de Petri é uma técnica que foi desenvolvida especificamente para modelar sistemas com interação de componentes concorrentes. É um modelo formal e abstrato do fluxo de informação, representado através de um grafo que modela as propriedades estáticas do sistema e que, através de sua execução, a Rede de Petri permite também representar as propriedades dinâmicas. A simplicidade e a força de representação das Redes de Petri fazem dessa técnica uma excelente ferramenta para a especificação de sistemas concorrentes (Peterson, 1981).

Existem duas linhas de pesquisa relacionadas às Redes de Petri: uma que se concentra na representação e análise de sistemas considerando, principalmente, os aspectos de concorrência e conflito e outra, que tem por objetivo principal a análise dos sistemas modelados através de Redes de Petri, a fim de derivar as propriedades dos mesmos (Peterson, 1977).

Segundo Murata (1989), as Redes de Petri podem ser utilizadas em uma grande variedade de aplicações. Elas podem ser aplicadas a qualquer área ou sistema que possa ser descrito graficamente como um diagrama e que precise de algum mecanismo para representar atividades paralelas ou concorrentes. No entanto, é salientado que quanto mais geral for o modelo, menor a facilidade de

analisá-lo. O aspecto mais crítico na utilização das Redes de Petri é a complexidade do problema, pois os modelos baseados nessa técnica, mesmo para sistemas modestos, tendem a se tornar muito grandes para serem analisados.

Como foi mencionado no Capítulo 2, dentre os métodos disponíveis para análise das Redes de Petri, o único que pode ser aplicado a todas as classes de redes é a árvore de alcançabilidade, que corresponde essencialmente à enumeração de todas as marcações alcançáveis. Mesmo assim, a aplicação desse método fica restrita às redes pequenas, devido à complexidade da explosão do espaço de estados (Murata, 1989), caracterizando a dificuldade encontrada na atividade de teste e validação das especificações baseadas nessa técnica.

Salienta-se que para as Redes de Petri, não foram encontrados na literatura métodos que gerem seqüências de transições que possam ser utilizadas no teste dessas especificações (LNCS-935, 1995; LNCS-815, 1994), diferentemente do que acontece para as Máquinas de Estados Finitos, que possuem alguns métodos de geração de seqüências de teste propostos. Assim, no contexto das Redes de Petri, propõe-se a aplicação do critério Análise de Mutantes como uma forma de conduzir a atividade de teste, que pode ser feita de maneira incremental, pelas próprias características do critério, possibilitando a elaboração de seqüências de transições capazes de distinguir entre os comportamentos dos mutantes e da Rede de Petri original.

Como já foi mencionado nos capítulos anteriores, é extremamente relevante a definição do conjunto de operadores de mutação para a aplicação bem sucedida do critério Análise de Mutantes. Com base nos operadores de mutação é que são geradas as Redes de Petri mutantes que caracterizam os erros que se desejam explorar na especificação original sob teste. Conseqüentemente, os resultados obtidos do teste baseado na Análise de Mutantes serão tanto mais representativos para o teste em questão, quanto também os forem os operadores de mutação, em relação aos erros que eles retratam.

Embora não exista comunicação das Redes de Petri com o mundo exterior, as Máquinas de Estados Finitos podem ser representadas através dessa técnica. Existem várias maneiras de modelar as interações com o mundo exterior. Uma

delas é representar as entradas e as saídas das MEFs como *lugares* da rede; outra, é modelar as entradas e saídas através de *transições*. Os estados das MEFs são representados por lugares na Rede de Petri e as marcações simulam o seu comportamento (Peterson, 1981).

Considerando a primeira maneira de representação, isto é, entradas e saídas das MEFs modeladas como lugares da rede, uma única Rede de Petri pode ser usada para modelar um sistema de MEFs interconectadas. De acordo com essa correspondência, os erros de operação e de transferência (Chow, 1978) considerados no contexto de Máquinas de Estados Finitos, correspondem, nas Redes de Petri, a erros onde um dos arcos de saída de uma transição leva a um lugar errado (Bochmann et al, 1992).

Dessa forma, sendo necessária a definição do conjunto de operadores de mutação para que o critério Análise de Mutantes pudesse ser aplicado no contexto de Redes de Petri, essa definição foi inspirada na classificação de erros, proposta por Chow (1978), para Máquinas de Estados Finitos.

Definido então o conjunto de operadores de mutação, o critério Análise de Mutantes foi aplicado a uma especificação baseada em Redes de Petri, que corresponde a um protocolo de comunicação, dado que nessa área, as Redes de Petri são também bastante utilizadas, assim como o são as Máquinas de Estados Finitos. Da mesma forma como foi conduzido para as Máquinas de Estados Finitos, os critérios de mutação alternativa foram aplicados no contexto de Redes de Petri para avaliar o uso dessa estratégia nesta técnica de especificação. A aplicação do critério foi feita através de um experimento realizado manualmente e seus resultados são apresentados neste capítulo.

O capítulo está assim organizado: na Seção 5.2 apresentam-se os principais conceitos das Redes de Petri. Na Seção 5.3 apresenta-se o conjunto de operadores de mutação proposto para essa técnica. Na Seção 5.4 mostra-se um exemplo de aplicação do critério Análise de Mutantes na validação de especificações baseadas em Redes de Petri, com base no conjunto de operadores definido e utilizando-se uma especificação de um protocolo de comunicação. Na Seção 5.5 apresenta-se a aplicação dos critérios de mutação alternativa ao

mesmo exemplo considerado na seção anterior e, na Seção 5.6 apresentam-se as considerações finais.

5.2 Redes de Petri

Rede de Petri (RP) (Peterson, 1981) é uma técnica usada para a modelagem de sistemas, constituída de dois componentes básicos: um conjunto P, de *lugares* e um conjunto T, de *transições*. O relacionamento entre esses dois componentes é dado através de duas funções que conectam as transições aos lugares: a função I, de *input* e a função O, de *output*. I define, para cada transição t_j , o conjunto $I(t_j)$ de *lugares de entrada*. O define, para cada transição t_j , o conjunto $O(t_j)$ de *lugares de saída* para a transição.

Esses quatro itens definem a estrutura de uma Rede de Petri. Formalmente, uma Rede de Petri é definida como uma quádrupla C, onde:

$C = (P, T, I, O)$

$P = \{ p_1, ..., p_n \}$ $n \geq 0$ (finito)

$T = \{ t_1, ..., t_m \}$ $m \geq 0$ (finito) $\qquad P \cap T = \phi$

$I = T \rightarrow P\infty$

$O = T \rightarrow P\infty$

A Rede de Petri pode ser representada graficamente, onde um *círculo* representa um lugar e uma *barra* representa uma transição. As funções de entrada e saída são representadas por arcos direcionados dos lugares para as transições e das transições para os lugares, respectivamente.

A execução de uma Rede de Petri é controlada pela posição e movimentação de marcas na rede, que são chamadas *tokens*. Os tokens são representados por pequenos círculos pretos que "residem" nos círculos correspondentes aos lugares da rede e são movidos pelo *disparo* das transições. A transição deve estar *habilitada* para que possa disparar e isso acontece quando todos os seus lugares de entrada possuem tokens. A transição dispara removendo

os tokens de seus lugares de entrada, gerando novos tokens que são depositados em todos os seus lugares de saída. A distribuição dos tokens na Rede de Petri define o *estado* da rede, que é chamada *marcação*. A marcação é denotada pelo vetor $\mu = (\mu_1, \mu_2,...,\mu_n)$, que fornece, para cada lugar da rede, o número de tokens nesse lugar, ou seja, $\mu (p_i) = \mu_i$.

5.3 Definição dos Operadores de Mutação para Redes de Petri

No projeto dos operadores de mutação para Máquinas de Estados Finitos tomou-se por base a classificação dos erros sugerida por Chow (1978). Essa classificação, como foi mencionado no capítulo anterior, considera M e M' Máquinas de Estados Finitos, onde M é o modelo correto e M' uma possível implementação, e define os erros da seguinte forma: *erros de operação* (quando M' não é equivalente a M mas pode se tornar equivalente a M, mudando-se apenas a função de saída de M'), *erros de transferência* (quando M' não é equivalente a M mas pode se tornar equivalente a M, mudando-se apenas a função de próximo estado de M') e *erros de estados extras ou ausentes* (quando, a fim de tornar M' equivalente a M, o número de estados em M' deve ser reduzido ou aumentado; como M' e M são minimais, um número diferente de estados implica que M' e M não são equivalentes).

No projeto dos operadores de mutação para Redes de Petri procurou-se tomar por base a classificação sugerida por Chow (1978) para MEFs, apesar de não existir uma correspondência direta. Por exemplo, um erro de transferência numa Rede de Petri poderia estar associado à função de saída da mesma. Assim, poderiam ser estabelecidos os operadores *input/output-faltando*, *input/output-extra*, *input/output-trocado*, *input/output-deslocado*, em relação a uma dada transição. Por exemplo, considerando o protocolo da Figura 5.1, o mutante retratado pela Figura 5.2a, reflete um erro de especificação sintetizado pelo operador *input-faltando*. Esse erro, do ponto de vista semântico da aplicação, corresponde a possibilitar a emissão de um novo quadro de mensagens sem o

recebimento prévio do "acknowledgement" relativo à transmissão de um quadro anterior. Assim, com base nessas diretrizes, estabeleceu-se um conjunto de operadores de mutação para Redes de Petri, que são definidos formalmente a seguir.

Da mesma forma como foi feito para Máquinas de Estados Finitos, apresentam-se, a seguir, os operadores de mutação para Redes de Petri, agrupados pela estrutura da rede que é alterada pela aplicação do operador, que neste caso pode ser: input, output e marcação.

- **Erros de Input**

Operador 1: *input faltando* (InpDel)

Este operador exclui, um por vez, os lugares de entrada de cada transição da rede.
Mutantes M_r, $0 \leq r \leq (m \cdot n)$, são gerados da seguinte maneira:

Para toda $t_j \in T$, define-se um mutante onde

$|I(t_j)_r| = (|I(t_j)| - 1)$

$I(t_j)_r = I(t_j) - \{p_i\}$, para cada $p_i \in I(t_j)$

Operador 2: *input extra* (InpIns)

Este operador inclui no conjunto de lugares de entrada de cada transição da rede, cada um dos lugares da rede que não consta originalmente desse conjunto.

Mutantes M_r, $0 \leq r \leq (m \cdot (n - 1))$, são gerados da seguinte maneira:

Para toda $t_j \in T$, define-se um mutante onde

$| I(t_j)_r | = (| I(t_j) | + 1)$

$I(t_j)_r = I(t_j) \cup \{ p_i \}$, para cada p_i tal que $p_i \in P$ e $p_i \notin I(t_j)$

Operador 3: *input deslocado* (InpShift)

Este operador, para cada transição da rede, exclui um lugar do conjunto de lugares de entrada da transição e inclui esse lugar no conjunto de lugares de entrada de cada uma das outras transições que não o possuísse originalmente.

Mutantes M_r, $0 \leq r \leq (m . n . (m - 1))$, são gerados da seguinte maneira

Para cada $t_j \in T$ definem-se $(| I(t_j) | . (m - 1))$ mutantes onde

$| I(t_j)_r | = | I(t_j) - 1 |$, onde $I(t_j)_r = I(t_j) - \{ p_k \}$ e

$| I(t_i)_r | = | I(t_i) + 1 |$, com

$I(t_i)_r = I(t_i) \cup \{ p_k \}$, para todo $i \neq j$ tal que $p_k \notin I(t_i)$

Operador 4: *input trocado* (InpAlt)

Este operador, para cada transição da rede, exclui um lugar do conjunto de lugares de entrada dessa transição e inclui ao conjunto cada um dos lugares da rede que não conste originalmente desse conjunto.

Mutantes M_r, $0 \leq r \leq (m . (n^2 / 4))$, são gerados da seguinte maneira:

Para cada $t_j \in T$ definem-se $| I(t_j) | . (n - | I(t_j) |)$ mutantes onde

$| I(t_j)_r | = | I(t_j) |$

$I(t_j)_r = I(t_j) - \{ p_i \} \cup \{ p_k \}$, para cada $p_i \in I(t_j)$ e
para cada $p_k \in P$ e $p_k \notin I(t_j)$

- **Erros de Output**

Operador 5: *output faltando* (OutDel)

Este operador exclui, um por vez, os lugares de saída de cada transição da rede.

Mutantes M_r, $0 \leq r \leq (m . n)$, são gerados da seguinte maneira:

Para cada $t_j \in T$ definem-se $| O(t_j) |$ mutantes onde

$| O(t_j)_r | = (| O(t_j) | - 1)$ onde

$O(t_j)_r = O(t_j) - \{ p_i \}$, para cada $p_i \in O(t_j)$

Operador 6: *output extra* (OutIns)

Este operador inclui no conjunto de lugares de saída de cada transição da rede, cada um dos lugares da rede que não constam originalmente desse conjunto.

Mutantes M_r, $0 \leq r \leq (m . n)$, são gerados da seguinte maneira:

Para cada $t_j \in T$ definem-se $(n - | O(t_j) |)$ mutantes onde

$| O(t_j)_r | = (| O(t_j) | + 1)$

$O(t_j)_r = O(t_j) \cup \{ p_i \}$, para cada p_i tal que $p_i \in P$ e $p_i \notin O(t_j)$

Operador 7: *output deslocado* (OutShift)

Este operador, para cada transição da rede, exclui um lugar do conjunto de lugares de saída da transição e inclui esse lugar no conjunto de lugares de saída de cada uma das outras transições que não o possuísse originalmente.

Mutantes M_r, $0 \leq r \leq (m . n . (m - 1))$, são gerados da seguinte maneira:

Para cada $t_j \in T$ definem-se $(| O (t_j) | . (m - 1))$ mutantes onde

$| O (t_j)_r | = | O (t_j) - 1 |$

$O (t_j)_r = O (t_j) - \{ p_k \}$ e

$| O (t_i)_r | = | O (t_i) + 1 |$, com

$O (t_i)_r = O (t_i) \cup \{ p_k \}$, para todo $i \neq j$ tal que $p_k \notin O (t_i)$

Operador 8: *output trocado* (OutAlt)

Este operador, para cada transição da rede, exclui um lugar do conjunto de lugares de saída dessa transição e inclui ao conjunto cada um dos lugares da rede que não conste originalmente desse conjunto.

Mutantes M_r, $0 \leq r \leq (m . (n^2 / 4))$, são gerados da seguinte maneira:

Para cada $t_j \in T$ definem-se $(| O (t_j) | . (n - | O (t_j) |))$ mutantes onde

$| O (t_j)_r | = | O (t_j) |$

$O (t_j)_r = O (t_j) - \{ p_i \} \cup \{ p_k \}$ para cada $p_i \in O (t_j)$ e
para cada $p_k \in P$ e $p_k \notin O (t_j)$

- **Erro de Marcação**

Operador 9: *alteração da marcação inicial* (MarkAlt)

Este operador altera a marcação inicial da rede, considerando as três alternativas descritas a seguir:

Seja a marcação inicial $\mu^0 = (\mu_1 , \mu_2 , ..., \mu_n)$ onde $\mu_i = \mu (p_i)$
Definem-se os mutantes das seguintes maneiras:

a) Acrescenta-se à marcação inicial cada um dos lugares da rede que não compõe essa marcação.

Seja $P' = \{ p_j \in P$ tal que $\mu (p_j) = 0 \}$ e $| P' | = k$
Mutantes M_r, $0 \leq r \leq k$, são gerados da seguinte maneira
Para cada $p_j \in P'$ gera-se um mutante onde
$\mu^0_r = (\mu_{1r} , \mu_{2r}, ..., \mu_{nr})$ e $\mu_{ir} = \mu (p_i)$, para $i \neq j$ e $\mu_{jr} = 1$

b) Retira-se da marcação inicial cada um dos lugares da rede que compõe essa marcação.

Seja $P'' = \{ p_j \in P$ tal que $\mu (p_j) = 1 \}$ e $| P'' | = s$
Mutantes M_r, $0 \leq r \leq s$, são gerados da seguinte maneira:
Para cada $p_j \in P''$ gera-se um mutante onde
$\mu^0_r = (\mu_{1r} , \mu_{2r}, ..., \mu_{nr})$ onde $\mu_{ir} = \mu (p_i)$, para $i \neq j$ e $\mu_{jr} = 0$

c) Retira-se da marcação inicial cada lugar da rede que compõe essa marcação e acrescenta-se cada um dos outros lugares que não a compõe.

Mutantes M_r, $0 \leq r \leq (k . s)$, são gerados da seguinte maneira:
Para cada (p_i, p_j), $p_i \in P''$ e $p_j \in P'$, define-se um mutante onde
$\mu^0_r = (\mu_{1r} , \mu_{2r}, ..., \mu_{nr})$ e $\mu_{qr} = \mu(p_q)$, para $q \neq i$ e $q \neq j$ e
$\mu_{ir} = \mu(p_j) = 0$ e $\mu_{jr} = \mu(p_i) = 1$

5.4 Um Exemplo de Aplicação do Critério Análise de Mutantes em Redes de Petri

Com base no conjunto de operadores de mutação definidos na seção anterior, aplicou-se o critério Análise de Mutantes no contexto de especificações baseadas em Redes de Petri.

A seguir, na Figura 5.1, apresenta-se uma Rede de Petri que especifica um Protocolo Nível 3 (Tanenbaum, 1989), a qual é utilizada como exemplo para explorar as idéias apresentadas neste trabalho.

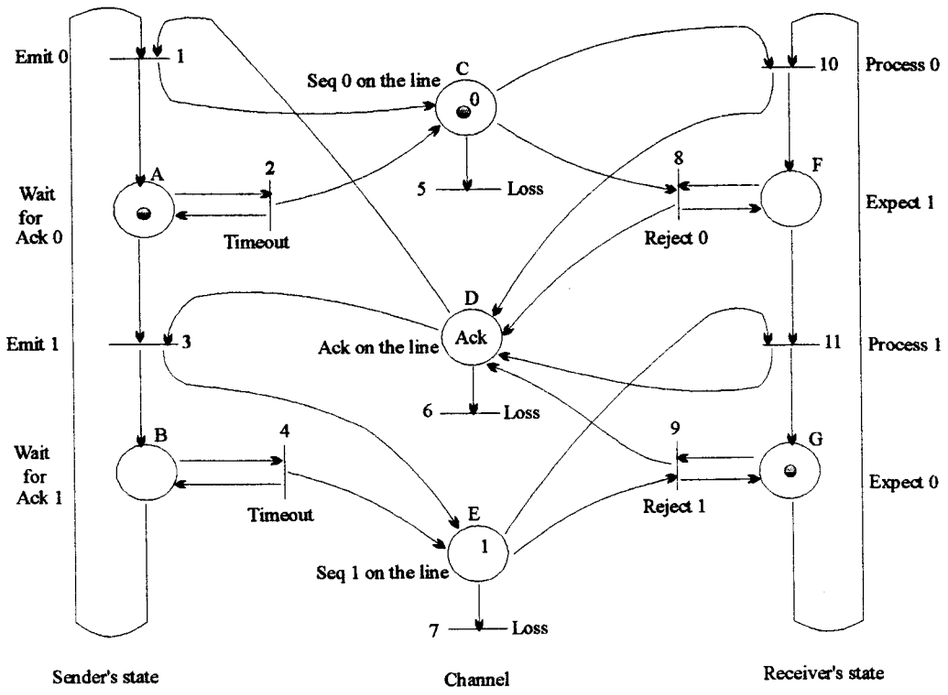


Figura 5.1 - Modelagem do Protocolo Nível 3 através de uma Rede de Petri (Tanenbaum, 1989)

Na Figura 5.2 apresentam-se exemplos da aplicação desses operadores sobre a Rede de Petri apresentada na Figura 5.1. Observa-se na Figura 5.2 que: em (a) excluiu-se D de I (3); em (b) excluiu-se A de I (3) e incluiu-se A em I (4); em (c) incluiu-se A em O (6); e em (d) excluiu-se D e incluiu-se B em O (9) .

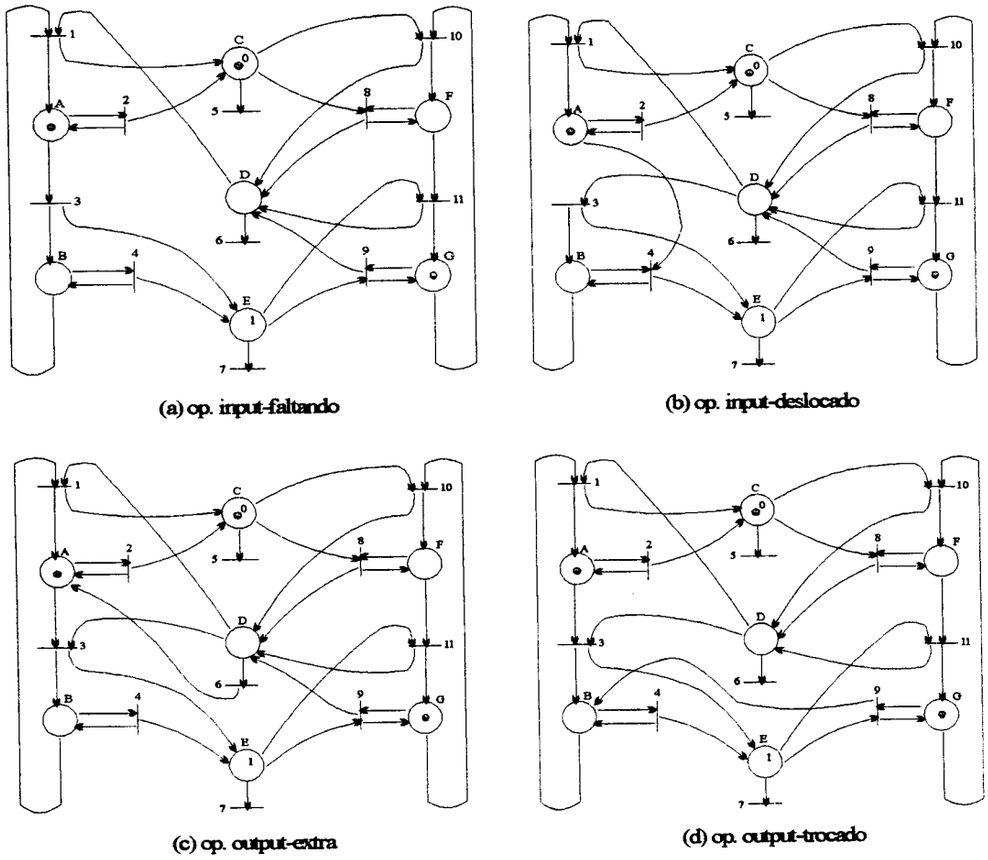


Figura 5.2 - Exemplos da Aplicação dos Operadores de Mutação para a Rede de Petri da Figura 5.1

Com base nos operadores definidos, a partir do Protocolo Nível 3, da Figura 5.1, foram geradas 643 Redes de Petri mutantes, conforme pode ser observado na Tabela 5.1. Não foi aplicado o operador *input deslocado* às transições 5, 6 e 7, pois eram gerados mutantes com transição sem lugares de entrada.

Inicialmente, determinou-se o conjunto *T-inic*, de seqüências de teste caracterizado por uma seqüência normal de operação, sem erros de transmissão e por uma seqüência que além de refletir problemas de comunicação, provocando a perda e retransmissão de quadros de mensagens e de "ack", garante também o disparo de todas as transições da Rede de Petri considerada.

T-inic ={ (10, 3, 11, 1), (10, 3, 11, 1, 2, 5, 10, 2, 6, 8, 3, 7, 4, 11, 4, 9) }

Para essas seqüências o modelo comportou-se de acordo com as expectativas, ou seja, refletiu o comportamento esperado do sistema a ser implementado. A adequação de *T-inic* em relação à Análise de Mutantes, considerando o conjunto de operadores de mutação definido na seção anterior, pode ser extraída da Tabela 5.1. Com os dados desta Tabela obtém-se que 13.84% dos mutantes permanecem vivos (um escore de mutação de 86.16%), ou porque as seqüências utilizadas não são adequadas para distinguir o comportamento da rede inicial em relação à rede mutante, ou porque a rede e o mutante em questão são equivalentes, ou seja, têm o mesmo comportamento para qualquer seqüência possível de execução da rede. No primeiro caso equivaleria dizer que se fosse tomado o mutante como a especificação "correta", as seqüências utilizadas não seriam sensíveis o suficiente para revelar esse tipo de erro. Assim, com base no conjunto de mutantes vivos, a massa de teste pode ser aprimorada levando a uma atividade de validação de melhor qualidade. Para exemplificar, considere o mutante da Figura 5.2b gerado pela aplicação do operador *input deslocado*. Para "matar" este mutante seria necessário inserir, por exemplo, a seqüência de teste (2, 5, 2, 10, 6, 2, 8, 3, 7, 4, 11, 6, 4, 9, 1), que simula seqüências de eventos que caracterizam canais de comunicação de péssima qualidade, o que implicaria na transmissão e retransmissão de quadros de mensagens e de "acknowledgments".

Para identificar se um determinado mutante apresenta um comportamento distinto do comportamento da Rede de Petri original verificou-se a marcação apresentada pelo mutante e pela Rede de Petri após a execução da seqüência de teste utilizada, sem diferenciar entre um ou mais tokens presentes nos lugares, ou seja, múltiplos tokens em um determinado lugar é equivalente à ocorrência de um único token. Assim, para que uma marcação μ seja distinguida de uma marcação

μ' é necessário que para algum lugar p_i, $\mu(p_i) = 0$ e $\mu'(p_i) \neq 0$, ou vice-versa. Esta abordagem poderia ser reformulada se fosse requerido que a Rede de Petri em questão apresentasse uma propriedade específica ou pertencesse a uma dada classe de redes, como por exemplo, "safe nets" ou "k-bounded nets". Assim, os mutantes que apresentassem uma marcação final ou mesmo intermediária que não preenchesse tais características, poderiam ser considerados mortos.

A comparação das marcações somente após a execução de toda a seqüência de teste, sem diferenciar um ou mais tokens e sem utilizar marcações intermediárias, afeta o escore de mutação e dificulta a eliminação dos mutantes, o que pode favorecer o aprimoramento da massa de teste e, conseqüentemente, o aumento de qualidade dessa atividade. Por exemplo, considere a seqüência s = (10, 3, 11, 1) de *T-inic* que ocorre no início da outra seqüência definida em *T-inic*; à primeira vista o testador poderia pensar em excluir s de *T-inic*. Neste caso, teria obtido um escore de mutação inferior, de 78.7% (21.3% dos mutantes permaneceriam vivos), o que implicaria em uma maior possibilidade de se aceitar como certa uma especificação errônea, pois o número de mutantes que teriam permanecido vivos teria sido maior. A massa de teste não teria sido sensível o suficiente para diferenciar esses mutantes, ou melhor dizendo, revelar os tipos de erros sintetizados por eles.

Tabela 5.1 - Síntese dos Resultados Obtidos da Aplicação da Análise de Mutantes ao Exemplo da Figura 5.1, com o Conjunto de Teste T-inic

Operador	*Total*	*Mortos*	*Vivos*
Op.1 input faltando	17	13	4
Op.2 input extra	60	59	1
Op.3 input deslocado	140	113	27
Op.4 input trocado	90	89	1
Op.5 output faltando	16	15	1
Op.6 output extra	61	41	20
Op.7 output deslocado	160	126	34
Op.8 output trocado	80	79	1
Op.9 alteração da marcação inicial	19	19	0
Total	643	554	89

O conjunto *T-inic* de seqüências de teste utilizado para a aplicação da Análise de Mutantes foi aprimorado procurando refletir aspectos semânticos da aplicação considerada, identificando-se seqüências e situações típicas: seqüências de transmissão com retransmissão do pacote número 0; seqüências de transmissão com retransmissão do pacote número 1; e seqüências de transmissão com problemas nos canais de transmissão e recepção, levando a retransmissões dos pacotes e respectivos quadros de "acknowledgment". Assim, a partir de *T-inic* obtém-se o conjunto *T-fim* constituído pelas seguintes seqüências típicas:

T-fim = { (10, 3, 11, 1), (10, 3, 11, 1, 2, 5, 10, 2, 6, 8, 3, 7, 4, 11, 4, 9),
(2, 5, 10, 3, 11, 1), (10, 6, 2, 8, 3, 11, 1), (10, 3, 4, 7, 11, 1),
(10, 3, 11, 6, 4, 9, 1), (2, 5, 10, 3, 4, 7, 11, 1),
(10, 6, 2, 8, 3, 11, 6, 4, 9, 1), (2, 5, 2, 10, 6, 2, 8, 3, 7, 4, 11, 6, 4, 9, 1) }

Com o conjunto de seqüências *T-fim*, mais completo e refletindo ocorrências típicas da natureza da aplicação considerada, obtém-se, com os dados da Tabela 5.2, um escore de mutação perto de 95%, ou seja, apenas 5.44% dos mutantes (35) permanecem vivos. Assim, em princípio, dever-se-ia acrescentar ao conjunto *T-fim* outras seqüências com o propósito de "matar" os mutantes que permaneceram vivos, se possível, pois como estabelecido anteriormente, entre eles pode haver mutantes que sejam equivalentes à Rede de Petri original.

A questão de equivalência é um ponto relevante para a aplicação mais efetiva e automatização do critério Análise de Mutantes. No escopo deste trabalho este aspecto não foi considerado e deverá ser abordado em trabalhos futuros.

Tabela 5.2 - Síntese dos Resultados Obtidos da Aplicação da Análise de Mutantes ao Exemplo da Figura 5.1, com o Conjunto de Teste T-fim

Operador	*Total*	*Mortos*	*Vivos*
Op.1 input faltando	17	15	2
Op.2 input extra	60	60	0
Op.3 input deslocado	140	127	13
Op.4 input trocado	90	90	0
Op.5 output faltando	16	16	0
Op.6 output extra	61	52	9
Op.7 output deslocado	160	149	11
Op.8 output trocado	80	80	0
Op.9 alteração da marcação inicial	19	19	0
Total	643	608	35

Fica assim evidente que o uso da Análise de Mutantes no contexto de Redes de Petri, proposto neste trabalho, fornece informações adicionais à validação de especificações baseadas nessa técnica, as quais complementam as atividades realizadas e as informações obtidas decorrentes do uso de técnicas como simulação e análise de alcançabilidade, entre outras.

5.5 Aplicação da Análise de Mutantes Alternativa em Redes de Petri

Nesta seção apresenta-se uma síntese dos resultados da aplicação dos critérios alternativos da Análise de Mutantes ao exemplo considerado na seção anterior. A Mutação Aleatória (n%) considerada foi a mutação 10% e a Mutação Restrita foi aplicada considerando-se dois conjuntos de operadores distintos: um deles, formado pelos operadores *(input/output)-faltando* e referenciado, quando necessário, pelo sufixo "2op" por conter apenas dois operadores; o outro, formado pelos operadores *(input/output)-faltando* e *(input/output)-trocado*, referenciado pelo sufixo "4op", por ser formado por 4 operadores de mutação.

No exemplo considerado utilizou-se apenas um conjunto de teste adequado para cada um dos critérios alternativos aplicados. Inicialmente, analisou-se a adequação das seqüências *T-inic* e *T-fim* em relação aos mutantes alternativos e completaram-se esses conjuntos a fim de se obter conjuntos de teste adequados para cada um desses critérios. Esses conjuntos estão apresentados na Tabela 5.3.

Tabela 5.3 - Conjuntos de Teste Adequados para os Critérios de Mutação Alternativa

	mutação 10%		input/output faltando		input/output faltando/trocado	
Conjuntos de teste / *Seqüências*	T-inic-10%	T-fim-10%	T-inic-2op	T-fim-2op	T-inic-4op	T-fim-4op
(10, 3 11, 1)	x	x	x	x	x	x
(10, 3, 11, 1, 2, 5, 10, 2, 6, 8, 3, 7, 4, 11, 4, 9)	x	x	x	x	x	x
(2, 5, 10, 3, 11, 1)		x		x		x
(10, 6, 2, 8, 3, 11, 1)	x	x	x	x	x	x
(10, 3, 4, 7, 11, 1)		x	x	x	x	x
(10, 3, 11, 6, 4, 9, 1)	x	x	x	x	x	x
(2, 5, 10, 3, 4, 7, 11, 1)		x		x		x
(10, 6, 2, 8, 3, 11, 6, 4, 9, 1)	x	x		x		x
(2, 5, 2, 10, 6, 2, 8, 3, 7, 4, 11, 6, 4, 9, 1)		x		x		x
(10, 8)	x	x				
(10, 2, 10)	x	x	x	x	x	x
(10, 3, 9, 1)			x	x	x	x

A seguir, analisa-se a adequação de cada conjunto alternativo adequado em relação ao critério Análise de Mutantes e aos outros critérios alternativos. Os resultados obtidos estão sumarizados na Tabela 5.4. Por exemplo, a mutação alternativa restrita (4op) correspondente à seleção dos operadores *(input/output)-faltando* e *(input/output)-trocado* requer 203 mutantes; *T-inic* é 0.9656 adequada em relação à mutação de *(input/output)-(faltando/trocado)*; *T-fim* é 0.9902

adequada em relação à mutação de *(input/output)-(faltando/trocado)*; *T-inic-2op* é 1 adequada em relação à mutação de *(input/output)-(faltando/trocado)* e à mutação restrita (2op) de *(input/output)-faltando*, mas é 0.9693 adequada em relação à mutação 10% e 0.9332 adequada em relação à mutação ("full mutation").

Tabela 5.4 - Dados de Análise do Fator "Strength"

critério / *(nº de mutantes)*		mutação 10%	input/output faltando	input/output faltando/ trocado	mutação
seqüência	*compr.*	(65)	(33)	(203)	(643)
T-inic	2	0.9077	0.8485	0.9656	0.8616
T-fim	9	0.9693	0.9394	0.9902	0.9456
T-inic-4op	7	0.9693	1	1	0.9332
T-fim-4op	11	0.9847	1	1	0.9643
T-inic-2op	7	0.9693	1	1	0.9332
T-fim-2op	11	0.9847	1	1	0.9643
T-inic-10%	7	1	0.9394	0.9902	0.9145
T-fim-10%	11	1	0.9697	0.9951	0.9643

Da Tabela 5.4 podem ser computadas as métricas C_1, C_2 e C_3 , apresentadas na Seção 3.3.1, do Capítulo 3, para cada um dos critérios de mutação alternativa utilizado. Esses resultados estão apresentados na Tabela 5.5. Como a métrica C_1 analisa o tamanho das seqüências adequadas aos critérios alternativos em relação ao tamanho da seqüência adequada à Análise de Mutantes, os 35 mutantes que permaneceram vivos com a aplicação de T-fim, foram analisados e novas seqüências foram adicionadas a esse conjunto para torná-lo adequado à Análise de Mutantes. Constam desse conjunto resultante, um total de 32 seqüências.

Tabela 5.5 - Dados de Custo e "Strength"

mutação / *métrica*		mutação 10%	input/output faltando	input/output faltando/trocado
custo	C_1	78.13%	78.13%	78.13%
	C_2	89.89%	94.87%	68.43%
strength	C_3	91.45%	93.32%	93.32%

Os resultados obtidos fornecem evidências de que os critérios de mutação alternativa levam a uma redução significativa de custo em termos do tamanho da seqüência de teste adequada e também do número de mutantes examinados. Por exemplo, considerando a mutação de *(input/output)-faltando*, a redução de custo em termos do tamanho da seqüência de teste adequada é de 78.13%; a redução de custo em termos do número de mutantes examinados é de 94.87% e a habilidade de distinguir mutantes não equivalentes é de 93.32%.

Como já foi mencionado anteriormente, devido ao escopo limitado do experimento, nenhuma observação conclusiva pode ser estabelecida, mas existem evidências de que os mesmos benefícios de se usar mutação alternativa no contexto de teste de programas, como mostra Wong et al (1994b), também pode ser obtido no contexto de Redes de Petri.

- tanto a mutação 10% como a mutação restrita fornecem uma redução significativa de custo em termos do tamanho da seqüência de teste e do número de mutantes examinados;
- os benefícios na redução do custo estão vinculados a uma pequena perda da força na habilidade de distinguir mutantes não equivalentes.

Desta forma, dependendo dos requisitos da aplicação, examinar apenas uma pequena porcentagem dos mutantes pode constituir uma heurística bastante útil na aplicação do teste de mutação. Esta abordagem também pode ser usada no estabelecimento de estratégias de teste incremental, como é proposto por Wong et al (1994a).

5.6 Considerações Finais

Apresentou-se neste capítulo a aplicação do critério Análise de Mutantes na validação de especificações baseadas em Redes de Petri. Da mesma forma como foi feito para Máquinas de Estados Finitos, para ser possível a aplicação do critério, fez-se necessária a definição de um conjunto de operadores de mutação para as Redes de Petri.

Esse conjunto de operadores de mutação foi inspirado na classificação de erros proposta por Chow (1978), para Máquinas de Estados Finitos. Embora não exista uma correspondência direta entre as duas técnicas, toda Máquina de Estados Finitos pode ser representada por uma Rede de Petri. Assim, com base na proposta de Chow (1978), procurou-se identificar, nas Redes de Petri, erros capazes de caracterizar aquela classificação.

Com base no conjunto de operadores definido e utilizando-se como exemplo a especificação de um protocolo de comunicação, aplicou-se o critério Análise de Mutantes no contexto de Redes de Petri. Como não se tem conhecimento de métodos para geração de seqüências de teste para Redes de Petri, definiu-se um conjunto inicial de seqüências para avaliar o comportamento dos mutantes. Como vários mutantes permaneceram vivos, esse conjunto inicial foi acrescido de outras seqüências de disparos de transições, com a finalidade de matá-los. Novamente, com esse outro conjunto, ainda permaneceram vivos alguns mutantes e o procedimento poderia ser repetido na tentativa de obter-se um escore de mutação de 100% (Fabbri et al, 1994c).

Os critérios de mutação alternativa também foram aplicados na validação de Redes de Petri e os resultados obtidos mostram uma redução de custo significativa, vinculada a uma pequena perda na eficácia do critério, de forma semelhante ao que foi obtido para Máquinas de Estados Finitos e ao que vem sendo obtido no teste de programas (Fabbri et al, 1995a).

Assim, mostrou-se neste capítulo, que a aplicação do critério Análise de Mutantes no contexto de Redes de Petri é uma forma de conduzir as atividades de teste de uma maneira mais sistemática e direcionada para os tipos de erros que se

desejam revelar. Mostrou-se também, através do exemplo, como conduzir essas atividades através de uma abordagem incremental, que é propiciada pelo próprio critério Análise de Mutantes, onde o conjunto de teste é acrescido de novas seqüências até que se consiga uma qualidade satisfatória para o teste que está sendo conduzido.

A evolução deste trabalho dar-se-á na direção de se refinar e complementar o conjunto de operadores de mutação através de estudos teóricos e empíricos que visem a caracterizar o custo e a eficácia da Análise de Mutantes no contexto de Redes de Petri. Nesse sentido, a definição e implementação de uma ferramenta de apoio é fundamental. No Capítulo 7 são discutidos os principais aspectos relacionados ao desenvolvimento de uma ferramenta desse gênero para Redes de Petri, com base na especificação da ferramenta Proteum-RS/FSM.

CAPÍTULO 6

ANÁLISE DE MUTANTES APLICADA EM STATECHARTS

6.1 Considerações Iniciais

A técnica Statecharts, proposta por Harel (1987a), é uma extensão das Máquinas de Estados Finitos, que procura justamente solucionar vários aspectos desta técnica que dificultam sua aplicação em sistemas complexos. Essa extensão corresponde à hierarquia dos estados, à capacidade de especificar paralelismo, ao mecanismo de comunicação através de "broadcasting", além de um conjunto de notações especiais que fornecem um maior poder de representação em relação às MEFs, possibilitando que sistemas complexos sejam descritos de forma bastante concisa. Por outro lado, como comentam Masiero et al (1994a), esse poder de representação proporcionado pela técnica Statecharts, torna mais difícil a atividade de validação das especificações em relação ao comportamento esperado.

Como foi mencionado no Capítulo 2, existem algumas formas de validação de especificações baseadas em Statecharts, sendo que algumas delas encontram-se disponíveis no ambiente StatSim (Masiero et al, 1991). No entanto, pelo fato desta técnica possuir uma série de recursos adicionais, nem sempre é possível aplicar-se essas formas de validação devido à especificação conter aspectos não tratados por elas, como é o caso da árvore de alcançabilidade, que não admite que sejam utilizadas variáveis nas condições dos eventos (Masiero et al, 1994a).

Sendo assim, muitas vezes torna-se difícil a identificação de determinados erros cometidos em uma especificação Statecharts. Nesse sentido, a aplicação da Análise de Mutantes no contexto de Statecharts vem complementar as formas de validação existentes, uma vez que, em geral, sempre são gerados mutantes que necessitam da elaboração de seqüências específicas de eventos, de forma que submetidos a elas, seus comportamentos sejam então distinguidos do comportamento da especificação original.

Neste capítulo definem-se três estratégias para testar especificações baseadas em Statecharts através do critério Análise de Mutantes: a Básica, a baseada em Ortogonalidade e a baseada em História. Essas estratégias estão fundamentadas no aspecto de decomposição, onde os componentes do Statecharts vão sendo evidenciados de acordo com os níveis de decomposição ou de hierarquia dos estados que são abstraídos pelos seus passos. A estratégia Básica tem por objetivo abstrair os componentes tipo OR do Statecharts, explorando o fato deles serem equivalentes a Máquinas de Estados Finitos. A Estratégia Baseada em Ortogonalidade abstrai os componentes tipo AND, com o objetivo de explorar aspectos de comunicação e sincronização. A Estratégia Baseada em História abstrai os componentes que possuem essa característica, com o objetivo de explorar aspectos relativos à retomada de contexto durante a simulação do modelo. Identificados esses componentes, aplicam-se a eles os operadores de mutação definidos para Statecharts, escolhendo-se aqueles que melhor explorem os aspectos abstraídos pelas estratégias e que melhor caracterizem os erros que se desejam explorar na especificação em teste.

Além das estratégias e dos operadores de mutação apresenta-se a proposta de aplicação dos critérios de mutação alternativa – Mutação Restrita e Mutação Aleatória – os quais podem ser combinados com as três estratégias, fornecendo várias alternativas para se atacar o teste de especificações baseadas em Statecharts, facilitando a exploração de determinados aspectos do sistema, e adequando-se a atividade de teste aos recursos de tempo e custo disponíveis.

O capítulo está organizado da seguinte maneira: apresentam-se, na Seção 6.2, através de um exemplo, a utilização de alguns recursos próprios da técnica Statecharts, bem como algumas definições da sintaxe dessa técnica, que

são essenciais para o entendimento das estratégias de teste propostas. Na Seção 6.3 comentam-se as vantagens obtidas na atividade de teste de Statecharts ao utilizar-se o critério Análise de Mutantes. Em seguida, na Seção 6.4, apresentam-se as estratégias de teste e os operadores de mutação definidos neste trabalho. Esses operadores podem ser subdivididos basicamente em três subconjuntos: 1) operadores que exploram aspectos relativos a Máquinas de Estados Finitos; 2) operadores que exploram aspectos das Máquinas de Estados Finitos Estendidas; e 3) operadores que abordam características intrínsecas à técnica Statecharts, como broadcasting, história e paralelismo. Ainda nessa seção, são propostos os critérios de mutação alternativa. Na Seção 6.5 são apresentadas as considerações finais.

6.2 Statecharts

Nesta seção apresentam-se exemplos que ilustram a utilização de alguns conceitos relacionados à técnica Statecharts, que foram mencionados no Capítulo 2. Em seguida, na Seção 6.2.1, apresenta-se um subconjunto da sintaxe associada a essa técnica, subconjunto este, necessário para o entendimento das estratégias e definições propostas neste capítulo.

Como já mencionado, Statecharts é uma técnica de especificação do aspecto comportamental de Sistemas Reativos proposta por Harel (1987a, 1987b), e que pode ser resumida da seguinte maneira:

Statecharts = diagramas de estado + decomposição +
ortogonalidade + broadcasting

Na Figura 6.1, apresentada adiante, tem-se a especificação de um relógio digital através da técnica Statecharts (Harel, 1987a), onde estão ilustrados vários dos conceitos apresentados no Capítulo 2. Uma breve descrição desse exemplo pode ser dada da seguinte maneira: o relógio *Citzen_Quartz_Multi_Alarm* possui um visor, uma campainha de dois tons e quatro botões de controle denotados por *a*, *b*, *c* e *d* . Ele pode mostrar a hora (com am/pm ou no modo 24 horas), ele tem

uma campainha que soa a toda hora, se estiver habilitada, dois alarmes independentes, um cronômetro de 1/100 de segundos (com marcador regular e de voltas), uma luz para iluminação, um indicador da carga da bateria e um dispositivo de teste da campainha. Algumas observações relevantes para entendimento do exemplo são:

- os principais eventos externos correspondem a apertar e soltar os botões (denotado, por exemplo, por *a* e *â* , respectivamente);
- T1 e T2 correspondem às horas marcadas para os alarmes e T corresponde à hora corrente;
- P1 corresponde à condição:
  alarm1_status.enabled $\wedge$ (alarm2_status.disabled $\vee$ T1 $\neq$ T2);
- P2 corresponde à condição:
  alarm2_status.enabled $\wedge$ (alarm1_status.disabled $\vee$ T1 $\neq$ T2);
- P corresponde à condição:
  alarm1_status.enabled $\wedge$ alarm2_status.enabled $\wedge$ T1 = T2;
- existe um ciclo que alterna os mostradores e que ocorre ao apertar-se repetidamente o botão *a*;
- os mostradores de hora e data estão relacionados pelo botão *d* mas o mostrador de horas volta a ficar ativo depois que o mostrador de data esteve ativo por 2 minutos;
- o estado que representa os mostradores (*displays*) possui capacidade para atualização dos dois alarmes, bem como da data e hora;
- os modos (subestados) de atualização são alcançados pelo botão *c*, sendo que a atualização da data e hora requer que se aperte o botão continuamente por 2 segundos; em qualquer um dos casos, apertando-se o botão *b* volta-se ao mostrador anterior;
- para os modos de atualização existe uma saída adicional, através do botão *c*, que não se aplica ao modo de atualização como um todo;
- apertar o botão *d* em qualquer um dos subestados de atualização causa uma saída e uma volta imediata ao subestado visitado mais recentemente, que é o próprio que foi deixado; esse ato é responsável pela atualização propriamente;
- o estado cronômetro (*stopwatch*) possui os subestados *regular* e *lap*, que são dois tipos diferentes de mostradores, e o estado *zero* que é um estado especial no qual o cronômetro está desligado e na posição inicial.

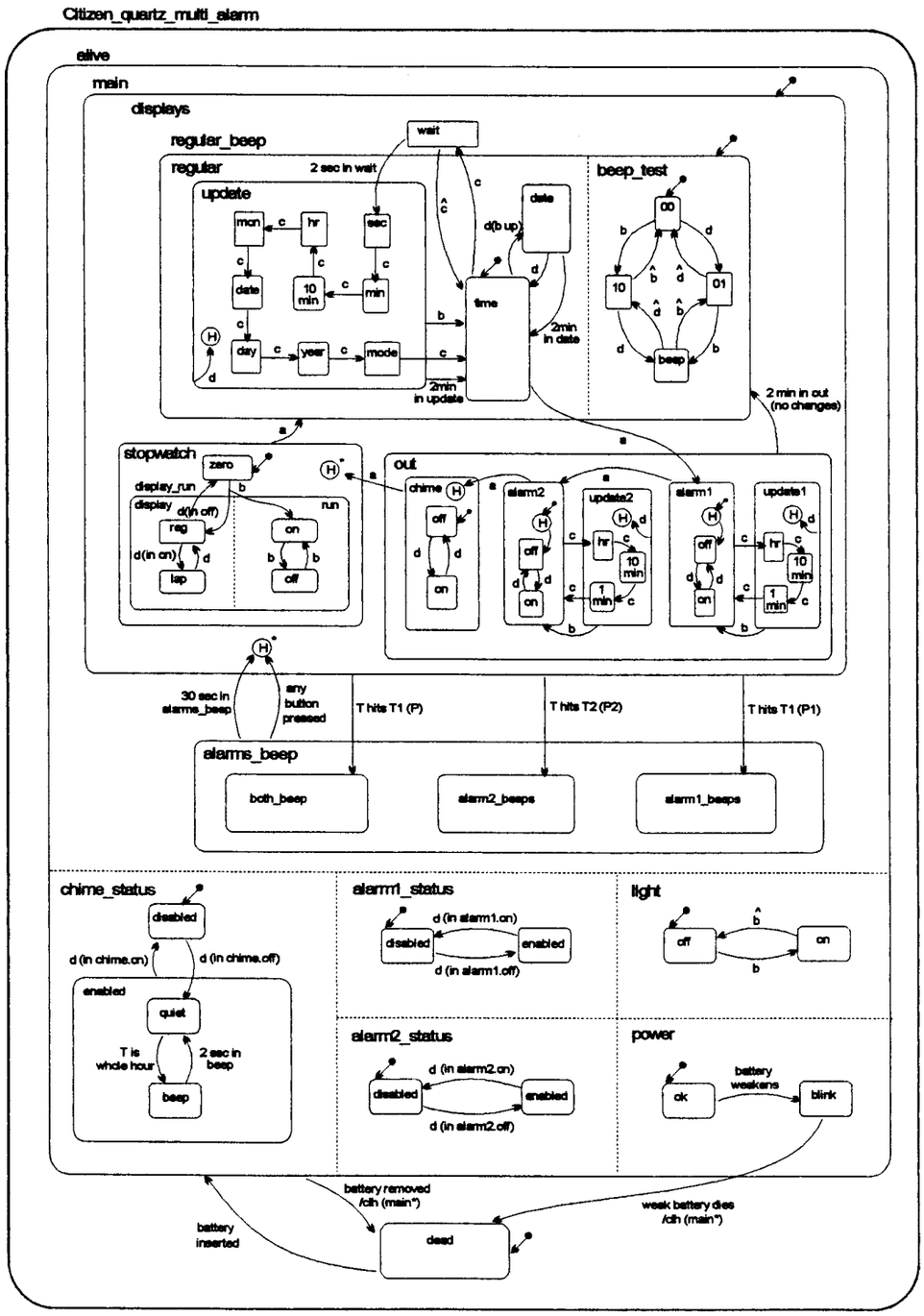


Figura 6.1 - Especificação de um Relógio Digital através da Técnica Statecharts (Harel, 1987a)

Com base na Figura 6.1, a idéia de *decomposição* pode ser observada através do estado *out* que é composto dos estados *chime*, *alarm2*, *update2*, *alarm1* e *update1*; a transição que tem por origem o estado *out* e por destino o estado *regular-beep* representa o fato de que ocorrido o evento *2_min_in_out(no changes)*, e disparada a transição, o estado *out* será abandonado e será ativado o estado *regular_beep*, independentemente de qual estado estiver ativo.

A *ortogonalidade*, que corresponde à especificação de estados do tipo AND, pode ser exemplificada pelo estado *alive*, da Figura 6.1, que é composto pelos estados paralelos: *chime_status*, *alarm1_status*, *alarm2_status*, *light*, *power* e *main*; graficamente, os estados paralelos são separados por linhas tracejadas.

Exemplo de aspectos de sincronização, representados por algumas condições como *in(s)* ou *ny(e)*, e por alguns eventos como *ex(s)* ou *en(s)*, pode ser dado pela condição *(in on)* da transição que tem como origem o subestado *reg* do estado *display*, denotado por *display.reg*, e como destino o estado *display.lap*. Essa condição faz com que essa transição seja disparada somente se no estado paralelo, isto é, no estado *run*, estiver ativo o estado *on*; caso contrário, essa transição não ocorre.

A capacidade do Statecharts de "lembrar" alguns estados que foram visitados previamente, isto é, o aspecto de *história*, pode também ser observado no Statecharts da Figura 6.1, na transição que tem por origem o estado *out.alarm2* e por destino o estado *chime*. A história *H* faz com que fique ativado o subestado de *chime* que esteve ativado por último na vez anterior em que esse estado foi visitado. Como exemplo de *H** , tem-se esse símbolo associado à transição que tem por origem o estado *out.chime* e por destino o estado *stopwatch*. Nesse caso, além de ser ativado ou o estado *zero* ou o estado *display_run*, dependendo do último que foi ativado na vez anterior, essa mesma regra vale para os subestados de *display_run*, tornando-se ativos os últimos visitados tanto no componente *display*, como no componente *run*.

Assim, foram exemplificadas as principais características intrínsecas à técnica Statecharts, que possibilitam um maior poder de representação em relação às Máquinas de Estados Finitos.

6.2.1 Sintaxe e Definições Básicas da Técnica Statecharts

Nesta seção apresentam-se um subconjunto da sintaxe e algumas definições básicas da técnica Statecharts (Harel, 1987b), necessárias ao entendimento dos conceitos e proposições introduzidos neste trabalho.

A sintaxe de Statecharts é definida a partir dos conjuntos básicos: S, de estados, T, de transições, E_p, de eventos primitivos, C_p, de condições primitivas e V_p, de variáveis. Usando esses conjuntos básicos, definem-se os conjuntos estendidos de: condições, expressões e rótulos (labels), assim como as inter-relações que os conectam.

a) ***Estados***: o conjunto de estados, *S*, é definido junto com uma função hierárquica, ρ, uma função tipo, ψ, um conjunto de símbolos de história *H* e uma função default δ.

A ***Função Hierárquica*** $\rho : S \rightarrow 2^S$ define para cada estado os seus subestados. Tem-se que se $\rho(x) = \rho(y)$ então $x = y$.

Existe um único estado $r \in S$ tal que $\forall s \in S \; r \notin \rho(s)$; *r* é a raiz do Statecharts.

ρ^* e ρ^+ são extensões de ρ definidas por: $\rho^*(s) = \cup \rho^i(s)$, $i \geq 0$

$\rho^+(s) = \cup \rho^i(s)$, $i \geq 1$.

A ***Função Tipo*** $\psi : S \rightarrow \{AND, OR\}$ define para cada estado o seu tipo.

Se $\rho(s) \neq \phi$ e $\psi(s) = OR$ então $\rho(s)$ é uma decomposição *xor* de *s* . Isso significa que quando um sistema está no estado *s* ele está em um e somente um dos subestados de *s*.

Se $\rho(s) \neq \phi$ e $\psi(s) = AND$ então $\rho(s)$ é uma decomposição *and* de s. Isso significa que quando um sistema está no estado s ele está simultaneamente em todos os subestados de s.

O conjunto de ***Símbolos de História***, H, está relacionado ao conjunto de estados pela função $\gamma: H \rightarrow S$ tal que:

$\gamma(h1) = \gamma(h2)$ implica $h1 = h2$ e $\gamma(H)$ é um subconjunto do conjunto de estados *OR*, ou seja, somente um estado *OR* pode ter um símbolo de história associado a ele.

Define-se $\omega: S \cup H \rightarrow S$ por $\omega(z)$ é z se $z \in S$ e $\omega(z)$ é $\gamma(z)$ se $z \in H$.

A ***Função Default*** $\delta: S \rightarrow 2^{S \cup H}$, define para um estado s, um conjunto de estados e símbolos de história que estão contidos no estado.

Se $x \in \delta(s)$ então: se $x \in S$, $x \in \rho^{+}(s)$,
se $x \in H$, $\gamma(x) \in \rho^{*}(s)$.

$\delta(s)$ é o conjunto default para s.

b) *Expressões*: o conjunto de variáveis é denotado por V_p. O conjunto de expressões, V, é definido indutivamente como segue:

1. Se k é um número então $k \in V$.
2. Se $v \in V_p$ então $v \in V$.
3. Se $v \in V$ então *current* $(v) \in V$.
4. Se $v_1, v_2 \in V$ e op é uma operação algébrica então $op(v_1, v_2) \in V$.

c) Condições: o conjunto de condições primitivas é denotado por C_p. O conjunto de condições, C, é definido indutivamente como segue:

1. $T, F \in C$; T, F correspondem a *true* e *false*, respectivamente.
2. Se $c \in C_p$ então $c \in C$.
3. Se $s \in S$ então $in(s) \in C$.
4. Se $e \in E$ então $not\_yet(e) \in C$.
5. Se $u, v \in V$, $R \in \{=, >, <, \neq, \geq, \leq\}$ então $u R v \in C$.
6. Se $c \in C$ então $current(c) \in C$.
7. Se $c_1, c_2 \in C$ então $c_1 \vee c_2, c_1 \wedge c_2, \neg c_1 \in C$.

d) Eventos: o conjunto de eventos primitivos é denotado por E_p. O conjunto de eventos, E, é definido indutivamente, como segue:

1. $\lambda \in E$; λ é o *evento nulo*.
2. Se $e \in E_p$ então $e \in E$.
3. Se $c \in C$ então $true(c), false(c) \in E$.
4. Se $v \in V$ então $changed(v) \in E$.
5. Se $s \in S$ então $exit(s), entered(s) \in E$.
6. Se $e_1, e_2 \in E$ então $e_1 \vee e_2, e_1 \wedge e_2 \in E$.
7. Se $e \in E, c \in C$ então $e[c] \in E$.

e é ***atômico*** se e é da forma 1-5

e) Ações: O conjunto de ações, A, é definido indutivamente como segue:

1. $\mu \in A$, μ é a *ação nula*.
2. Se $c \in C_p, d \in C$ então $c := d \in A$.
3. Se $v \in V_p, u \in V$ então $v := u \in A$.
4. Se $a_i \in A$, i = 0, ..., n então $a_0 ; \ldots ; a_n \in A$.

$a \in A$ é ***atômica*** se ela é da forma 1-3.

f) Rótulos: O conjunto de *rótulos*, L, é o conjunto de pares $E \times A$, e para $l = (e, a)$, $e \in E$, $a \in A$, escreve-se *e/a*. Informalmente, se *e/a* é o rótulo de uma transição t, então t é ativada por e e a é executada quando t é disparada.

g) Transições: O conjunto de *transições*, T, é definido como o conjunto de triplas $T \subset 2^S \times L \times 2^{S \cup H}$. A transição $t = (X, l, Y)$ é composta de um conjunto *origem* X, um conjunto *destino* Y, denotados por *source(t)*, *target(t)*, respectivamente, e um rótulo l. Informalmente, se $l = e/a$, se o sistema está em X e e ocorre, então t está habilitada e pode ser disparada. Se t é disparada, a é executada e o sistema vai então para Y.

Em seguida, estão alguns termos e conceitos usados na definição da semântica (Harel, 1987b), a qual não é apresentada neste trabalho. Os termos e conceitos apresentados a seguir são suficientes para o entendimento da estratégia de teste definida neste trabalho.

a) Um estado s é ***básico*** se $\rho(s) = \phi$

b) Para um conjunto de estados X, o ***Menor Ancestral Comum de X***, denotado por ***LCA(X)*** é o estado x definido como segue: $LCA(X) = x$ se e somente se:

1. $X \subseteq \rho^*(x)$
2. $\forall s \in S$, $X \subseteq \rho^*(s) \Rightarrow x \in \rho^*(s)$

Para um conjunto de estados X, o ***Menor Ancestral Estrito Comum OR de X***, denotado por $\boldsymbol{LCA^+(X)}$ é o estado x definido como segue: $LCA^+(X) = x$ se e somente se:

1. $X \subseteq \rho^+(x)$
2. $\psi(x) = OR$
3. $\forall s \in S$ se $\psi(s) = OR$ então $X \subseteq \rho^*(s) \Rightarrow x \in \rho^*(s)$

6.3 A Análise de Mutantes como Forma Complementar de Teste de Statecharts

Nesta seção, apresenta-se, através de um exemplo, como o critério Análise de Mutantes pode ser aplicado de forma complementar na atividade de teste de especificações baseadas em Statecharts.

O Statecharts utilizado como exemplo, apresentado na Figura 6.2, foi extraído de Cangussu et al (1995), onde são discutidos aspectos de validação de Statecharts através da execução programada. Essa especificação retrata um controle de processos, onde existem dois processadores, CONTROL_PROCESSOR_ONE e CONTROL_PROCESSOR_TWO, os quais podem estar livres (FREE1 ou FREE2) ou ocupados (BUSY1 ou BUSY2), e um controlador de fila, CONTROL_QUEUE, que trata de controlar os processos que chegam e enfileirá-los, caso os processadores estejam ocupados. Os eventos que podem ocorrer são: *process*, que corresponde à chegada de um processo e *deliver1* ou *deliver2* que correspondem à liberação de um processador, quando este termina de atender um processo. Sempre que os processadores estão livres, ocorre o evento *lambda[cr(x>0)]/x:=x-1*, que se encarrega de verificar se a fila não está vazia para que um novo processo seja atendido, decrementando portanto, o número de processos na fila.

O procedimento empregado foi realizado com o apoio das funções disponíveis no ambiente StatSim (Masiero et al, 1991) e pode ser descrito da seguinte forma: utilizou-se um programa baseado na Linguagem de Controle de Execução (LCE) (Cangussu, 1993), semelhante àquele apresentado em Cangussu et al (1995), de onde obteve-se a seqüência de eventos gerados, aplicados à especificação durante a execução programada, bem como as configurações alcançadas pelo Statecharts em cada passo dessa execução. Na Figura 6.3 apresentam-se esses resultados, através do relatório produzido pela execução da LCE. Como pode ser visto, a seqüência gerada é composta pelos eventos (process, process, process, process $\wedge$ deliver2, etc.), apresentado no relatório pelo item "Events triggered" e as configurações alcançadas relativas à essa seqüência são: (QUEUE, FREE1, BUSY2), (QUEUE, BUSY1, BUSY2), (QUEUE,

BUSY1, BUSY2), (QUEUE, BUSY1, FREE2), etc., apresentadas no relatório pelo item "Configuration of Active States". Estipulou-se nesse programa um total de 30 passos ("Number of Simulated Steps"), sendo que na Figura 6.3 estão apresentados apenas os passos relacionados à seqüência de eventos mencionada.

Considerando-se como exemplo o "Step 1", as informações produzidas pelo relatório são: a configuração relativa ao passo, isto é, os estados que foram ativados (QUEUE, FREE1 e BUSY2); o evento que ocorreu (process) e a transição interna disparada (lambda[cr(x>0)]/x:=x-1); o estado que foi desativado (FREE2) e o estado que foi ativado (BUSY2). Além disso, são apresentados os valores das variáveis *x*, que corresponde ao número de processos na fila, *y* e *z* que controlam se os processadores estão ocupados (valor 1) ou desocupados (valor 0). No campo de mensagem, através de instruções da LCE, pode-se apresentar qualquer notificação que se considere relevante, como por exemplo, o número de processos na fila, como é apresentado no "Step 3" da Figura 6.3.

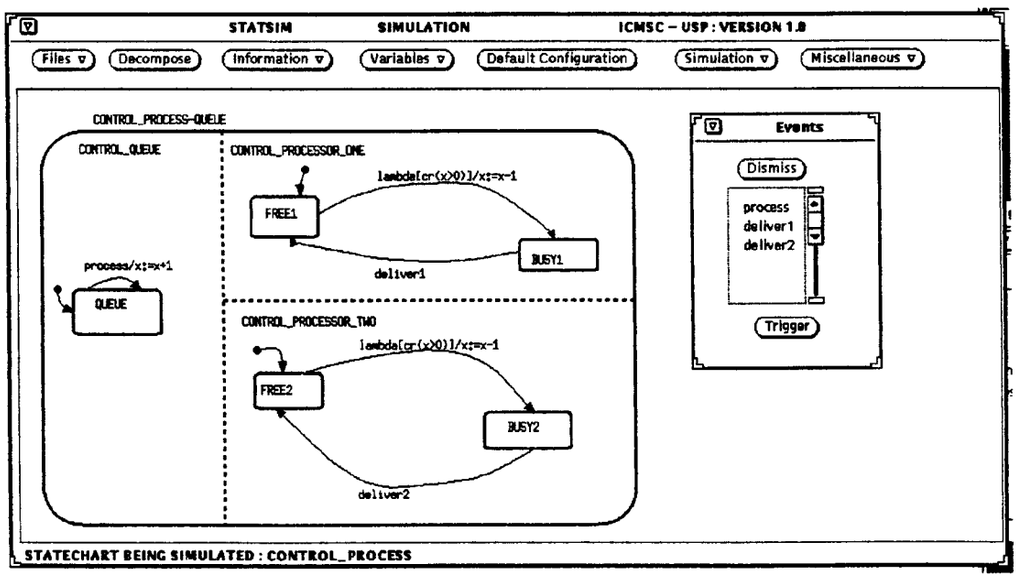


Figura 6.2 - Statecharts Relativo ao Sistema de Controle de Processos (Cangussu et al, 1995)

```
Statechart in Simulation: CONTROL_PROCESS
Number of Simulated Steps: 30
---------- Step 1 ----------

Configuration of Active States:

  CONTROL_PROCESS_QUEUE
    CONTROL_QUEUE
      QUEUE
    CONTROL_PROCESSOR_ONE
      FREE1
    CONTROL_PROCESSOR_TWO
      BUSY2

Events triggered     : process
Internal Transictions : lambda[cr(x>0)]/x:=x-1
States Turned On     : BUSY2
States Turned Off    : FREE2

Variable        Current Value      Average Value
  x                   0                0.0000
  y                   0                0.0000
  z                   1                0.2500

Messages :
-----------------------------------------
---------- Step 2 ----------

Configuration of Active States:

  CONTROL_PROCESS_QUEUE
    CONTROL_QUEUE
      QUEUE
    CONTROL_PROCESSOR_ONE
      BUSY1
    CONTROL_PROCESSOR_TWO
      BUSY2

Events triggered     : process
Internal Transictions : lambda[cr(x>0)]/x:=x-1
States Turned On     : BUSY1
States Turned Off    : FREE1

Variable        Current Value      Average Value
  x                   0                0.0000
  y                   1                0.1429
  z                   1                0.1429

Messages :
-----------------------------------------
---------- Step 3 ----------

Configuration of Active States:

  CONTROL_PROCESS_QUEUE
    CONTROL_QUEUE
      QUEUE
    CONTROL_PROCESSOR_ONE
      BUSY1
    CONTROL_PROCESSOR_TWO
      BUSY2

Events triggered     : process
Internal Transictions :
States Turned On     :
States Turned Off    :

Variable        Current Value      Average Value
  x                   1                0.1111
  y                   1                0.1111
  z                   1                0.1111

Messages : Queue Size = 1
-----------------------------------------
---------- Step 4 ----------

Configuration of Active States:

  CONTROL_PROCESS_QUEUE
    CONTROL_QUEUE
      QUEUE
    CONTROL_PROCESSOR_ONE
      BUSY1
    CONTROL_PROCESSOR_TWO
      FREE2

Events triggered     : process,deliver2
Internal Transictions :
States Turned On     : FREE2
States Turned Off    : BUSY2

Variable        Current Value      Average Value
  x                   2                0.3636
  y                   1                0.0909
  z                   0                0.0909

Messages :
-----------------------------------------
---------- Step 5 ----------

Configuration of Active States:

  CONTROL_PROCESS_QUEUE
    CONTROL_QUEUE
      QUEUE
    CONTROL_PROCESSOR_ONE
      BUSY1
    CONTROL_PROCESSOR_TWO
      BUSY2

Events triggered     :
Internal Transictions : lambda[cr(x>0)]/x:=x-1
States Turned On     : BUSY2
States Turned Off    : FREE2

Variable        Current Value      Average Value
  x                   1                0.4167
  y                   1                0.0833
  z                   1                0.1667

Messages :
-----------------------------------------
```

Figura 6.3 - Relatório Produzido pela Execução Programada do Exemplo da Figura 6.2

Em seguida, elaborou-se um mutante do Statecharts da Figura 6.2, gerado a partir da aplicação do operador de mutação *troca de constantes por constantes requeridas (TraConsAltCons)*, apresentado na próxima seção, e definido para explorar o uso de expressões aritméticas, aspecto este que é utilizado tanto em Máquinas de Estados Finitos Estendidas como em Statecharts. Esse mutante é apresentado na Figura 6.4 e como pode ser observado, a alteração ocorrida refere-se à substituição da constante "1", na transição que tem como origem o estado FREE1 e como destino o estado BUSY1, pelo valor "0". Essa alteração corresponde a não decrementar o número de processos que estão na fila, sempre que um processo é atendido pelo processador CONTROL_PROCESSOR_ONE. Com base na seqüência gerada pela execução programada, o mutante foi simulado através da simulação interativa, e alguns dos passos resultantes dessa simulação podem ser vistos também na Figura 6.4, através da janela "LOG".

O resultado obtido na simulação interativa foi exatamente o mesmo do obtido na execução programada gerando, a cada passo, a mesma configuração de estados em ambos os casos, fazendo com que o mutante produzisse o mesmo resultado do Statecharts original, isto é, permanecesse vivo. Isso corresponde a dizer que a seqüência de eventos gerada pela execução programada não seria adequada ao critério Análise de Mutantes, pois não distinguiria o comportamento de todos os mutantes. Embora o valor intermediário das variáveis pudessem ser verificados durante a simulação interativa – e nesse caso, o mutante poderia ser morto pois esses valores eram diferentes dos valores da execução programada – essa abordagem não foi adotada pois seria equivalente a estar-se aplicando a mutação fraca, usando-se estados intermediários da simulação para comparar a execução dos mutantes. A proposta feita no trabalho é que seja aplicada a mutação forte, observando-se a configuração final da especificação. Essa abordagem deixa a massa de teste mais robusta, isto é, com maior capacidade de detectar erros, produzindo, conseqüentemente, um produto de maior qualidade.

Observa-se que, para que o comportamento do mutante fosse distinguido, seria necessária a elaboração de uma seqüência de eventos obedecendo à seguinte ordem: *process*, *process*, *deliver1* (ou *deliver2*), ... , pois no Statecharts original, para que o processador que ficou livre, através do evento *deliver*, voltasse a ficar ocupado, seria necessária a ocorrência de um novo evento *process*,

enquanto que no mutante, o processador que ficou livre passaria novamente ao estado de ocupado uma vez que, erroneamente, haveria um processo na fila esperando para ser atendido.

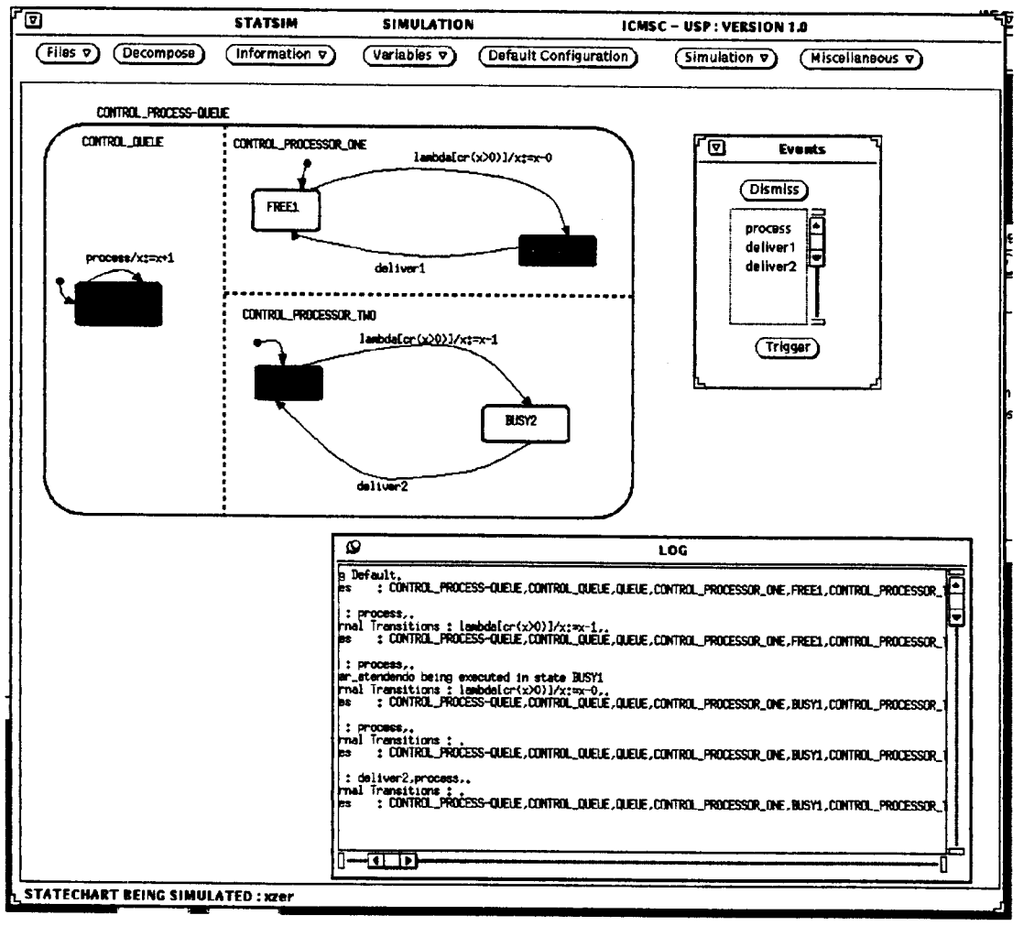


Figura 6.4 - Mutante do Statecharts da Figura 6.2 e Resultado Parcial da Simulação Interativa do Mutante (LOG)

Salienta-se que na execução programada, os eventos são gerados segundo uma distribuição probabilística. Assim, nada garante que uma determinada seqüência de eventos seja gerada por uma execução. Caso tivesse sido submetido o mutante à execução programada, utilizando-se o mesmo programa do exemplo, o erro poderia ter sido identificado ao analisar-se o relatório produzido pela execução. No entanto, salienta-se que o Statecharts considerado é bastante simples e faz uso de uma única variável, a variável *x*. Certamente, para um sistema complexo e que utilize um grande número de variáveis, a identificação do erro não seria simples, pois tudo dependeria da probabilidade da seqüência de eventos gerada exercitar todas as transições que possuissem variáveis associadas a elas. Fazendo uma analogia com o teste de programas, seria equivalente à aplicação do critério todos os caminhos, que como se sabe, é impraticável.

Além disso, testar-se uma especificação com o objetivo de garantir que ela não possui determinados tipos de erros, aqueles mais relevantes e críticos para a aplicação que esteja sendo modelada, não seria uma tarefa simples, considerando-se como técnica de apoio à atividade de teste a execução programada. Certamente, a aplicação do critério Análise de Mutantes atenderia a esse objetivo de forma mais eficiente, à medida que os operadores de mutação podem retratar os erros que se desejam detectar na especificação em teste.

Portanto, como foi mostrado através do exemplo, com a aplicação do critério Análise de Mutantes o testador seria forçado a elaborar seqüências de teste específicas, com o intuito de matar os mutantes que permanecem vivos, melhorando a qualidade da massa de teste e, conseqüentemente, aumentado a confiança na especificação que está sendo testada.

Tendo ilustrado que o critério Análise de Mutantes pode constituir uma forma complementar de teste em relação aos métodos propostos na literatura, nas seções seguintes apresentam-se as estratégias propostas neste trabalho, acompanhadas da definição de operadores de mutação para a técnica Statecharts.

6.4 Estratégias de Teste para a Técnica Statecharts

As três estratégias de teste propostas para a técnica Statecharts estão baseadas no aspecto de decomposição dessa técnica pois, através da abstração do Statecharts, selecionam, por nível de decomposição, os componentes pertencentes àquele nível. Dessa forma, elas possibilitam que a condução do teste seja realizada através das abordagens "top-down" ou "bottom-up". Identificados os componentes, o teste da especificação pode partir do nível mais alto de abstração, que corresponde ao próprio Statecharts, ou então, pode partir do nível mais baixo de abstração, o qual é composto pelos componentes cujos estados são todos átomos.

As estratégias, denominadas *Básica*, *Baseada em Ortogonalidade* e *Baseada em História*, diferenciam, essencialmente, no aspecto que se deseja explorar na atividade de teste. No entanto, todas elas aplicam o mesmo procedimento para abstrair os componentes de cada nível do Statecharts.

Um ponto importante do critério Análise de Mutantes é a análise dos mutantes que foram gerados. Assim, identificados os componentes, pode-se avaliar o comportamento dos mutantes gerados para eles com base em duas abordagens: 1) isoladamente, considerando-se o comportamento do componente original e do componente mutante, comparando-se a configuração alcançada por eles; 2) dentro do contexto do próprio Statecharts, comparando-se a configuração alcançada pelo Statecharts como um todo.

Em seguida, é apresentada cada uma das estratégias juntamente com um exemplo de sua aplicação, o qual está elaborado com base no Statecharts da Figura 6.1.

- **Estratégia Básica**

A estratégia Básica abstrai os componentes do tipo OR que correspondem essencialmente à estrutura básica de composição dos Statecharts que é a

Máquina de Estados Finitos. Esses componentes podem ser vistos como Máquinas de Estados Finitos Estendidas. O objetivo principal dessa estratégia é testar os componentes OR, abordando também aspectos de integração entre eles. Ela pode ser considerada como uma estratégia mais geral, onde não se exploram aspectos intrínsecos à técnica Statecharts.

Exemplificando o uso da estratégia, primeiramente constrói-se um conjunto formado pelos estados que não possuem decomposição, como por exemplo, os estados *dead*, *date*, *wait*, *light.off*, *power.ok*, *enabled.beep*, *chime_status.disabled*, *alarms_beep.alarm1beeps*, *update.mode*, dentre vários outros. Em seguida, é estabelecido um conjunto formado pelas MEFs que correspondem ao "zoom-out" desses estados, ou seja, que possuem esses estados como seus subestados. Como exemplo, pode-se citar as MEFs *light*, *power*, *enabled*, *alarms_beep*, *chime*, *run*, *beep_test* e *update*, dentre outras. Assim, definiu-se um primeiro nível de abstração de MEFs. O segundo nível de abstração corresponde ao "zoom-out" dessas MEFs do primeiro nível e é formado pelas MEFs *chime_status*, *stopwatch*, *regular* e *out*. O terceiro nível de abstração, é composto pela MEF *displays*. O quarto nível de abstração das MEFs é composto pelo estado *main* e, finalmente, o quinto nível é composto pela MEF *Citizen_quartz_multi_alarm*, que corresponde ao próprio Statecharts.

Sistematizando a estratégia de acordo com a notação apresentada na Seção 6.2.1, ela é descrita através de um algoritmo. Note-se que a cada iteração, define-se o conjunto dos estados candidatos a MEFs que compõem um determinado nível de abstração e que esse conjunto tende a diminuir à medida que o nível de abstração vai se tornando mais geral, ao ponto de corresponder ao próprio Statecharts, caso este seja do tipo OR. Se o Statecharts for do tipo AND esse conjunto torna-se vazio depois de abstraídos todos os componentes paralelos correspondentes às MEFs que compõem o Statecharts. Portanto, esse conjunto, denominado $CLCA^{+}_{i}$ define a condição de parada do algoritmo, o qual é apresentado em seguida:

Início Estratégia Básica

** B_1 é o conjunto de estados básicos **

$B_1 = \{ s \in S \mid \rho(s) = \phi \}$

** $CLCA^+_1$ é o conjunto de estados candidatos a MEFs do nível de abstração 1 **

$CLCA^+_1 = \{ s \in S \mid s = LCA^+ (b_i) , b_i \in B_1 \}$

** $CMEF_1$ é o conjunto das MEFs do nível de abstração 1 **

$CMEF_1 = \{ s \in CLCA^+_1 \mid \forall s_i \in \rho(s) , \rho(s_i) = \phi \}$

$i = 1$

Enquanto $\mid CLCA^+_i \mid > 1$ *faça*

$i = i + 1$

** B_i é o conjunto dos estados pertencentes às MEFs que ainda não foram abstraídas **

$B_i = B_{i-1} - \{ \cup \rho(s) , \text{para todo } s \in CMEF_{i-1} \}$

$- \{ \cup \rho(s_i) \mid \psi(s_i) = AND \wedge s_i \in \rho(s) , s \in CMEF_{i-1} \}$

$\cup CMEF_{i-1}$

** $CLCA^+_i$ é o conjunto dos estados candidatos a MEFs do nível de abstração i **

$CLCA^+_i = \{ s \in S \mid s = LCA^+ (b_i) , b_i \in B_i \}$

** $CMEF_i$ é o conjunto das MEFs do nível de abstração i **

$CMEF_i = \{ s \in CLCA^+_i \mid \forall s_i \in \rho(s) ,$

se $\psi(s_i) = OR$ então $s_i \in B_i$

c.c. $\forall s_j \in \rho(s_i) , s_j \in B_i \}$

fim-enquanto

Fim Estratégia Básica

Figura 6.5 - Estratégia Básica

Aplicando-se a estratégia proposta ao Statecharts da Figura 6.1, pode-se identificar as Máquinas de Estados Finitos Estendidas que compõem cada um dos níveis, como é apresentado em seguida:

$B_1 = \{ s \in S \mid \rho(s) = \phi \}$

$B_1 = \{$ dead, light.off, light.on, power.ok, power.blink, alarm1_status.disabled, alarm1_status.enabled, alarm2_status.disabled, alarm2_status.enabled, enabled.quiet, enabled.beep, chime_status.disabled, alarms_beep.alarm1beeps, alarms_beep.alarm2beeps, alarms_beep.bothbeep, update1.hr, update1.10min, update1.1min, alarm1.off, alarm1.on, update2.hr, update2.10min, update2.1min, alarm2.off, alarm2.on, chime.off, chime.on, run.on, run.off, display.reg, display.lap, zero, beep_test.00, beep_test.01, beep_test.10, beep_test.beep, update.sec, update.min, update.10min, update.hr, update.mon, update.date, update.day, update.year, update.mode, time, date, wait $\}$

$CLCA^+_1 = \{ s \in S \mid s = LCA^+(b_i) , b_i \in B_1 \}$

$CLCA^+_1 = \{$ citzen_quartz_multi_alarm, light, power, alarm1_status, alarm2_status, enabled, chime_status, alarms_beep, update1, alarm1, update2, alarm2, chime, run, display, stopwatch, beep_test, update, regular, displays $\}$

$CMEF_1 = \{ s \in CLCA^+_1 \mid \forall s_i \in \rho(s) , \rho(s_i) = \phi \}$

$CMEF_1 = \{$ light, power, alarm1_status, alarm2_status, enabled, alarms_beep, update1, alarm1, update2, alarm2, chime, run, display, beep_test, update $\}$

$B_2 = B_1 - \{ \cup \rho(s) , \text{para todo } s \in CMEF_1 \}$

$- \{ \cup \rho(s_i) \mid \psi(s_i) = AND \wedge s_i \in \rho(s) , s \in CMEF_1 \}$

$\cup CMEF_1$

$B_2 = \{$ dead, chime_status.disabled, zero, time, date, wait, light, power, alarm1_status, alarm2_status, enabled, alarms_beep, update1, alarm1, update2, alarm2, chime, run, display, beep_test, update $\}$

$CLCA^+_2 = \{ s \in S \mid s = LCA^+(b_i) , b_i \in B_2 \}$

$CLCA^+_2 = \{$ citzen_quartz_multi_alarm, chime_status, stopwatch, regular, displays, main, out $\}$

$CMEF_2 = \{ s \in CLCA^+_2 \mid \forall s_i \in \rho(s),$
se $\psi(s_i) = OR$ então $s_i \in B_2$
c.c. $\forall s_j \in \rho(s_i), s_j \in B_2 \}$
$CMEF_2 = \{$ chime_status, stopwatch, regular, out $\}$

$B_3 = B_2 - \{\cup \rho(s)$, para todo $s \in CMEF_2\}$
$- \{\cup \rho(s_i) \mid \psi(s_i) = AND \wedge s_i \in \rho(s), s \in CMEF_2\}$
$\cup CMEF_2$
$B_3 = \{$ dead, wait, light, power, alarm1_status, alarm2-status, alarms_beep,
beep_test, chime_status, stopwatch, regular, out $\}$

$CLCA^+_3 = \{ s \in S \mid s = LCA^+(b_i), b_i \in B_3 \}$
$CLCA^+_3 = \{$ citzen_quartz_multi_alarm, displays, main $\}$

$CMEF_3 = \{ s \in CLCA^+_3 \mid \forall s_i \in \rho(s),$
se $\psi(s_i) = OR$ então $s_i \in B_3$
c.c. $\forall s_j \in \rho(s_i), s_j \in B_3 \}$
$CMEF_3 = \{$ displays $\}$

$B_4 = B_3 - \{\cup \rho(s)$, para todo $s \in CMEF_3\}$
$- \{\cup \rho(s_i) \mid \psi(s_i) = AND \wedge s_i \in \rho(s), s \in CMEF_3\}$
$\cup CMEF_3$
$B_4 = \{$ dead, light, power, alarm1_status, alarm2_status, alarms_beep,
chime_status, displays $\}$

$CLCA^+_4 = \{ s \in S \mid s = LCA^+(b_i), b_i \in B_4 \}$
$CLCA^+_4 = \{$ citzen_quartz_multi_alarm, main $\}$

$CMEF_4 = \{ s \in CLCA^+_4 \mid \forall s_i \in \rho(s),$
se $\psi(s_i) = OR$ então $s_i \in B_4$
c.c. $\forall s_j \in \rho(s_i), s_j \in B_4 \}$
$CMEF_4 = \{$ main $\}$

$B_5 = B_4 - \{ \cup \rho(s) , \text{para todo } s \in CMEF_4 \}$

$- \{ \cup \rho(s_i) \mid \psi(s_i) = AND \wedge s_i \in \rho(s) , s \in CMEF_4 \}$

$\cup \ CMEF_4$

$B_5 = \{$ dead, light, power, alarm1_status, alarm2_status, chime_status, main $\}$

$CLCA^+_5 = \{ s \in S \mid s = LCA^+(b_i) , b_i \in B_5 \}$

$CLCA^+_5 = \{$ citzen_quartz_multi_alarm $\}$

$CMEF_5 = \{ s \in CLCA^+_5 \mid \forall s_i \in \rho(s) ,$

se $\psi(s_i) = OR$ então $s_i \in B_5$

c.c. $\forall s_j \in \rho(s_i) , s_j \in B_5 \}$

$CMEF_5 = \{$ citzen_quartz_multi_alarm $\}$

Observa-se então que existem cinco níveis de abstração, correspondentes aos conjuntos $CMEF_1$ a $CMEF_5$. Em cada um desses níveis identificam-se Máquinas de Estados Finitos Estendidas (MEFEs), com base nas quais os mutantes serão gerados ao serem aplicados os operadores de mutação definidos nas próximas seções.

- **Estratégia Baseada em Ortogonalidade**

Essa estratégia explora, na atividade de teste de um Statecharts, o aspecto de paralelismo. Nesse caso, são selecionados como alvos de mutação, em cada nível de abstração do Statecharts, apenas os componentes tipo AND que compõem a especificação. Na Figura 6.6, apresenta-se o algoritmo que extrai do Statecharts os componentes que satisfazem essa condição.

Como se pode observar, o algoritmo que apóia a Estratégia Baseada em Ortogonalidade está baseado no algoritmo da Figura 6.5, correspondente à Estratégia Básica. A diferença fundamental entre eles consiste em verificar, a cada iteração, se as MEFs abstraídas possuem subestados do tipo "AND".

Início Estratégia Ortogonalidade

** B_1 é o conjunto de estados básicos **

$B_1 = \{ s \in S \mid \rho(s) = \phi \}$

** $CLCA^+_1$ é o conjunto de estados candidatos a MEFs do nível de abstração 1 **

$CLCA^+_1 = \{ s \in S \mid s = LCA^+ (b_i) , b_i \in B_1 \}$

** $CMEF_1$ é o conjunto das MEFs do nível de abstração 1 **

$CMEF_1 = \{ s \in CLCA^+_1 \mid \forall s_i \in \rho(s) , \rho(s_i) = \phi \}$

** $Cort_1$ é o conjunto dos subestados AND das MEFs do nível de abstração 1 **

$Cort_1 = \{ s_i \mid s_i \in \rho(s) \wedge \psi(s_i) = AND , s \in CMEF_1 \}$

$i = 1$

Enquanto $\mid CLCA^+_i \mid > 1$ *faça*

$i = i + 1$

** B_i é o conjunto dos estados pertencentes às MEFs que ainda não foram abstraídas **

$B_i = B_{i-1} - \{ \cup \rho(s) , \text{para todo } s \in CMEF_{i-1} \}$

$- \{ \cup \rho(s_i) \mid \psi(s_i) = AND \wedge s_i \in \rho(s) , s \in CMEF_{i-1} \}$

$\cup CMEF_{i-1}$

** $CLCA^+_i$ é o conjunto dos estados candidatos a MEFs do nível de abstração i **

$CLCA^+_i = \{ s \in S \mid s = LCA^+ (b_i) , b_i \in B_i \}$

** $CMEF_i$ é o conjunto das MEFs do nível de abstração i **

$CMEF_i = \{ s \in CLCA^+_i \mid \forall s_i \in \rho(s) ,$

$\text{se } \psi(s_i) = OR \text{ então } s_i \in B_i$

$\text{c.c. } \forall s_j \in \rho(s_i) , s_j \in B_i \}$

** $Cort_i$ é o conjunto dos subestados AND das MEFs do nível de abstração i **

$Cort_i = \{ s_i \mid s_i \in \rho(s) \wedge \psi(s_i) = AND , s \in CMEF_i \}$

fim-enquanto

Fim Estratégia Ortogonalidade

Figura 6.6 - Estratégia Baseada em Ortogonalidade

Em seguida, considerando o Statecharts da Figura 6.1, apresenta-se um exemplo de aplicação do algoritmo que apóia a Estratégia Baseada em Ortogonalidade:

B_1 = { dead, light.off, light.on, power.ok, power.blink, alarm1_status.disabled, alarm1_status.enabled, alarm2_status.disabled, alarm2_status.enabled, enabled.quiet, enabled.beep, chime_status.disabled, alarms_beep.alarm1beeps, alarms_beep.alarm2beeps, alarms_beep.bothbeep, update1.hr, update1.10min, update1.1min, alarm1.off, alarm1.on, update2.hr, update2.10min, update2.1min, alarm2.off, alarm2.on, chime.off, chime.on, run.on, run.off, display.reg, display.lap, zero, beep_test.00, beep_test.01, beep_test.10, beep_test.beep, update.sec, update.min, update.10min, update.hr, update.mon, update.date, update.day, update.year, update.mode, time, date, wait }

$CLCA^+_1$ = { citzen_quartz_multi_alarm, light, power, alarm1_status, alarm2_status, enabled, chime_status, alarms_beep, update1, alarm1, update2, alarm2, chime, run, display, stopwatch, beep_test, update, regular, displays }

$CMEF_1$ = { light, power, alarm1_status, alarm2_status, enabled, alarms_beep, update1, alarm1, update2, alarm2, chime, run, display, beep_test, update }

$COrt_1 = \phi$

B_2 = { dead, chime_status.disabled, zero, time, date, wait, light, power, alarm1_status, alarm2_status, enabled, alarms_beep, update1, alarm1, update2, alarm2, chime, run, display, beep_test, update }

$CLCA^+_2$ = { citzen_quartz_multi_alarm, chime_status, stopwatch, regular, displays, main, out }

$CMEF_2$ = { chime_status, stopwatch, regular, out }

$COrt_2$ = { display_run }

B_3 = { dead, wait, light, power, alarm1_status, alarm2-status, alarms_beep, beep_test, chime_status, stopwatch, regular, out }

$CLCA^+_3$ = { citzen_quartz_multi_alarm, displays, main }

$CMEF_3$ = { displays }

$COrt_3$ = { regular_beep }

B_4 = { dead, light, power, alarm1_status, alarm2_status, alarms_beep, chime_status, displays }

$CLCA^+_4$ = { citzen_quartz_multi_alarm, main }

$CMEF_4$ = { main }

$COrt_4 = \phi$

B_5 = { dead, light, power, alarm1_status, alarm2_status, chime_status, main }

$CLCA^+_5$ = { citzen_quartz_multi_alarm }

$CMEF_5$ = { citzen_quartz_multi_alarm }

$COrt_5$ = { alive }

- **Estratégia Baseada em História**

Essa estratégia explora a característica de história, isto é, a capacidade da técnica Statecharts em memorizar a última configuração atingida por um estado. Uma vez escolhida essa característica, são selecionados como alvos de mutação apenas os componentes (as MEFs Estendidas) que possuem símbolos de história associados a eles. Assim, definidos esses componentes, o testador pode dar ênfase ao aspecto de história, para cada nível de abstração do Statecharts. Para obtenção dessas MEFs Estendidas, define-se o algoritmo da Figura 6.7:

Início Estratégia História

** B_1 é o conjunto de estados básicos **

$B_1 = \{ s \in S \mid \rho(s) = \phi \}$

** $CLCA^+_1$ é o conjunto de estados candidatos a MEFs do nível de abstração 1 **

$CLCA^+_1 = \{ s \in S \mid s = LCA^+ (b_i) , b_i \in B_1 \}$

** $CMEF_1$ é o conjunto das MEFs do nível de abstração 1 **

$CMEF_1 = \{ s \in CLCA^+_1 \mid \forall s_i \in \rho(s) , \rho(s_i) = \phi \}$

** $CMEF_{H1}$ é o conjunto das MEFs do nível de abstração 1 que possuem história **

$CMEF_{H1} = CMEF_1 \cap \gamma (H)$

$i = 1$

Enquanto $| CLCA^+_i | > 1$ *faça*

$i = i + 1$

** B_i é o conjunto dos estados pertencentes às MEFs que ainda não foram abstraídas **

$B_i = B_{i-1} - \{ \cup \rho(s) , \text{para todo } s \in CMEF_{i-1} \}$

$- \{ \cup \rho(s_i) \mid \psi(s_i) = AND \wedge s_i \in \rho(s) , s \in CMEF_{i-1} \}$

$\cup CMEF_{i-1}$

** $CLCA^+_i$ é o conjunto dos estados candidatos a MEFs do nível de abstração i **

$CLCA^+_i = \{ s \in S \mid s = LCA^+ (b_i) , b_i \in B_i \}$

** $CMEF_i$ é o conjunto das MEFs do nível de abstração i **

$CMEF_i = \{ s \in CLCA^+_i \mid \forall s_i \in \rho(s) ,$

$\text{se } \psi(s_i) = OR \text{ então } s_i \in B_i$

$\text{c.c. } \forall s_j \in \rho(s_i) , s_j \in B_i \}$

** $CMEF_{Hi}$ é o conjunto das MEFs do nível de abstração i que possuem história **

$CMEF_{Hi} = CMEF_i \cap \gamma (H)$

fim-enquanto

Fim Estratégia História

Figura 6.7 - Estratégia Baseada em História

Observa-se que para a aplicação dessa estratégia foi utilizado também o mesmo algoritmo da Estratégia Básica, apresentado na Figura 6.5, restringindo-se, no entanto, o conjunto de MEFEs selecionadas em cada nível de abstração do Statecharts. Essa restrição exclui do conjunto todas as MEFEs que não pertencem ao conjunto γ (H) , calculando a cada iteração, o conjunto $CMEF_{Hi} = CMEF_i \cap \gamma$ (H). Dessa forma, ficam selecionadas como alvos de mutação apenas as MEFEs que possuem o símbolo de história associado a elas.

Em seguida, ilustra-se a aplicação desse algoritmo no Statecharts da Figura 6.1. Primeiramente, deve ser estabelecido o conjunto γ (H) para que as MEFEs abstraídas em cada nível da especificação sejam restringidas àquelas que possuem a característica de história. Assim:

γ (H) = { update1, alarm1, update2, alarm2, chime, stopwatch, update, displays }

B_1 = { dead, light.off, light.on, power.ok, power.blink, alarm1_status.disabled, alarm1_status.enabled, alarm2_status.disabled, alarm2_status.enabled, enabled.quiet, enabled.beep, chime_status.disabled, alarms_beep.alarm1beeps, alarms_beep.alarm2beeps, alarms_beep.bothbeep, update1.hr, update1.10min, update1.1min, alarm1.off, alarm1.on, update2.hr, update2.10min, update2.1min, alarm2.off, alarm2.on, chime.off, chime.on, run.on, run.off, display.reg, display.lap, zero, beep_test.00, beep_test.01, beep_test.10, beep_test.beep, update.sec, update.min, update.10min, update.hr, update.mon, update.date, update.day, update.year, update.mode, time, date, wait }

$CLCA^+_1$ = { citzen_quartz_multi_alarm, light, power, alarm1_status, alarm2_status, enabled, chime_status, alarms_beep, update1, alarm1, update2, alarm2, chime, run, display, stopwatch, beep_test, update, regular, displays }

$CMEF_1$ = { light, power, alarm1_status, alarm2_status, enabled, alarms_beep, update1, alarm1, update2, alarm2, chime, run, display, beep_test, update }

$CMEF_{H1}$ = { update1, alarm1, update2, alarm2, chime, update }

B_2 = { dead, chime_status.disabled, zero, time, date, wait, light, power, alarm1_status, alarm2_status, enabled, alarms_beep, update1, alarm1, update2, alarm2, chime, run, display, beep_test, update }

$CLCA^+_2$ = { citzen_quartz_multi_alarm, chime_status, stopwatch, regular, displays, main, out }

$CMEF_2$ = { chime_status, stopwatch, regular, out }

$CMEF_{H2}$ = { stopwatch }

B_3 = { dead, wait, light, power, alarm1_status, alarm2-status, alarms_beep, beep_test, chime_status, stopwatch, regular, out }

$CLCA^+_3$ = { citzen_quartz_multi_alarm, displays, main }

$CMEF_3$ = { displays }

$CMEF_{H3}$ = { displays }

B_4 = { dead, light, power, alarm1_status, alarm2_status, alarms_beep, chime_status, displays }

$CLCA^+_4$ = { citzen_quartz_multi_alarm, main }

$CMEF_4$ = { main }

$CMEF_{H4} = \phi$

B_5 = { dead, light, power, alarm1_status, alarm2_status, chime_status, main }

$CLCA^+_5$ = { citzen_quartz_multi_alarm }

$CMEF_5$ = { citzen_quartz_multi_alarm }

$CMEF_{H5} = \phi$

Portanto, com base no algoritmo da Figura 6.5, correspondente à Estratégia Básica, foram construídos os algoritmos das Figuras 6.6 e 6.7, com o objetivo de explorar aspectos específicos da técnica Statecharts, por nível de abstração da especificação. Salienta-se que assim como na Estratégia Básica, os outros algoritmos possuem uma condição de parada pois a cada iteração, constróem-se conjuntos de componentes relativos aos níveis mais altos de abstração, até que se chegue ou no componente raiz, caso este seja do tipo OR, ou então não se tenha mais componentes para abstrair, caso o raiz seja do tipo AND. Em outras palavras, enquanto a cardinalidade do conjunto $CLCA^{+}_{i}$ for maior que um, ou seja, $| CLCA^{+}_{i} | > 1$. Uma vez selecionados os componentes, aplicam-se a eles os operadores de mutação de acordo com os tipos de erros que se desejam explorar na atividade de teste. Certamente, para as estratégias Baseada em Ortogonalidade e Baseada em História, os operadores mais releventes são aqueles que exploram aspectos de broadcasting e de história, respectivamente. No entanto, qualquer um dos operadores definidos para a técnica Statecharts, apresentados nas próximas seções, podem ser aplicados, a critério do testador.

6.4.1 Operadores de Mutação para Statecharts que Abordam Aspectos de Máquinas de Estados Finitos

Nesta seção apresentam-se os operadores de mutação que abordam os aspectos de Máquinas de Estados Finitos no contexto de especificações baseadas em Statecharts. Portanto, esses operadores são similares àqueles definidos no Capítulo 4, e aplicam-se em componentes do tipo OR de um Statecharts, que equivalem a Máquinas de Estados Finitos.

Observa-se que ao isolar-se um componente s, do tipo OR de um Statecharts, muitas vezes esse componente está relacionado a um outro componente s' através de transições que saem de s' ou de seus subestados e chegam em subestados de s . Os operadores definidos nesta seção consideram essas transições que conectam dois componentes OR distintos, aplicando mutações apenas nas transições que saem do componente s e vão para outros componentes. Dessa forma, as transições que chegam no componente s serão

consideradas quando, por exemplo, o componente s' for o alvo dos operadores. Ao serem consideradas essas transições está-se, de certa forma, tratando aspectos de validação em nível de integração dos componentes do Statecharts.

As considerações apresentadas em seguida têm o objetivo de restringir os elementos básicos de um Statecharts ao componente s , correspondente a um estado tipo OR , alvo dos operadores de mutação.

Sejam:

$s \in S$ e $|\rho(s)| = k$

$T|_s = \{ t \in T \mid t = (X, l, Y)$, $X \in \rho(s)$ e $Y \in \rho(s) \vee Y \in S$, $l = e/a$, $e \in E|_s$, $a \in A|_s \}$
ou seja, é o conjunto de transições que têm por origem um subestado do componente s.

$T|_{si,sj} = \{ t \in T|_s \mid t = (X, l, Y)$, onde $X = s_i$ e $Y = s_j$, $s_i \in \rho(s)$, $s_j \in \rho(s) \vee s_j \in S \}$
ou seja, é o conjunto de transições existentes do estado s_i para o estado s_j .

$E|_s = \{ e \in E \mid \exists t \in T$, $t = (X, l, Y)$, $l = e/a$, $X \in \rho(s)$ e $Y \in \rho(s) \vee Y \in S \}$, $|E|_s| = m$
ou seja, é o conjunto de eventos restritos ao componente s.

$E|_{si,sj} = \{ e \in E|_s \mid \exists t = (s_i, l, s_j)$, $l = e/a$, $a \in A|_s$, $s_i \in \rho(s)$, $s_j \in \rho(s) \vee s_j \in S \}$
ou seja, é o conjunto de todos os eventos que provocam transições de s_i para s_j.

$E|_{si,sjt} = \{ e' \in E|_{si,sj} \mid \exists t = (s_i, l, s_j)$, $l = e/a$ e e' compõe e , $s_i \in \rho(s)$, $s_j \in \rho(s) \vee s_j \in S \}$
ou seja, é o conjunto de eventos restritos a uma transição específica do estado s_i para o estado s_j.

$A|_s = \{ a \in A \mid \exists t \in T$, $t = (X, l, Y)$, $l = e/a$, $X \in \rho(s)$ e $Y \in \rho(s) \vee Y \in S \}$, $|A|_s| = n$
ou seja, é o conjunto de ações restritas ao componente s.

$A|_{si,sjlt} = \{ a \in A|_s \mid \exists t \in T,\ t = (s_i, l, s_j),\ l = e/a,\ s_i \in \rho(s),\ s_j \in \rho(s) \vee s_j \in S \}$

ou seja, é o conjunto das ações restritas a uma transição específica do estado s_i para o estado s_j.

$\delta(s) = s_0$ o estado default de s

Considere também:

nt_{ext} : o número de transições que saem do componente s e vão para um componente externo s' e

ns_{ext} : o número de estados que pertencem ao componente s' onde chega uma transição que sai do componente s.

Obs: no caso de Statecharts podem existir vários arcos de um estado s_i para um estado s_j e cada arco representa uma transição.

Operador 1: *alteração do estado default* (TraDefStaDel)

Este operador altera o estado default do componente s do Statecharts de forma que em cada mutante, cada um de seus outros subestados passa a ser o estado default.

Mutantes M_r, $0 \leq r \leq (k\text{-}1)$, são gerados da seguinte maneira:

$(E|_s)_r = E|_s$; $(A|_s)_r = A|_s$; $(T|_s)_r = T|_s$; $(\rho(s))_r = \rho(s)$;

$(\delta(s))_r = s_j$, $s_j \neq s_0$, para todo $s_j \in \rho(s)$, onde s_0 é o estado default inicial.

Operador 2: *arco faltando* (TraArcDel)

Este operador exclui um arco (transição) do componente s.

Mutantes M_r, $0 \leq r \leq ((m.k) + nt_{ext})$, são gerados da seguinte maneira:

Para cada par de estados (s_i , s_j) onde $s_i \in \rho(s)$ e $s_j \in \rho(s) \vee s_j \in S$ tal que $T|_{si,sj} \neq \phi$ geram-se mutantes onde:

$(E|_s)_r \subseteq E|_s$; $(A|_s)_r \subseteq A|_s$; $(\rho(s))_r = \rho(s)$; $(T|_s)_r \neq T|_s$;

$(T|_s)_r = T|_s - \{ t \}$ para cada $t \in T|_{si,sj}$

Operador 3: *evento faltando* (TraEveDel)

Este operador exclui cada um dos eventos associados a cada transição. Ou seja, se o evento for composto exclui-se um evento primitivo por vez. Se ele for primitivo, esse operador é equivalente ao operador arco faltando.

Mutantes M_r , $0 \leq r \leq ((m.k) + nt_{ext})$, são gerados da seguinte maneira:

Para cada par de estados (s_i , s_j) onde $s_i \in \rho(s)$ e $s_j \in \rho(s) \vee s_j \in S$ tal que $T|_{si,sj} \neq \phi$ geram-se mutantes onde:

$(E|_s)_r \subseteq E|_s$; $(A|_s)_r \subseteq A|_s$; $(\rho(s))_r = \rho(s)$; $(T|_s)_r \neq T|_s$;

se $| E|_{si,sjlt} | = 1$ faz-se: $(T|_{si,sj})_r = T|_{si,sj} - \{ t \}$

se $| E|_{si,sjlt} | > 1$ faz-se $(T|_{si,sj})_r = T|_{si,sj} - \{ t \} \cup \{ t_r \}$, onde

$t = (s_i , l , s_j)$, $l = e/a$, e é composto pelos elementos de $E|_{si,sjlt}$

$t_r = (s_i , l_r , s_j)$, $l_r = e_r /a$, e_r é composto pelos elementos de $E|_{si,sjlt} - \{ e_i \}$ excluindo-se, juntamente com e_i , o operador "associado" ao evento e_i , mantendo-se a ordem de prioridade dos operadores.

ou seja:

se e_i é da forma $e_1 \wedge (e_2 \vee e_3)$

excluir e_2 significa manter a expressão da seguinte forma: $e_1 \wedge e_3$

Operador 4: *evento extra* (TraEveIns)

Este operador inclui no conjunto de transições dos subestados do componente s que estejam conectados a outros estados, uma nova transição cujo evento pertence ao conjunto de eventos do componente s , mas não pertence ao conjunto de eventos do subestado.

Mutantes M_r , $0 \leq r \leq (k \,.\, (m - 1))$, são gerados da seguinte maneira:

Para cada par de estados (s_i , s_j) onde $s_i \in \rho(s)$ e $s_j \in \rho(s) \vee s_j \in S$ tal que $T|_{si,sj} \neq \phi$ geram-se $(m - | E|_{si,sj} |)$ mutantes onde:

$(E|_s)_r = E|_s$; $(A|_s)_r = A|_s$; $(\rho(s))_r = \rho(s)$; $(T|_s)_r \neq T|_s$;

$(T|_{si,sj})_r = T|_{si,sj} \cup \{ t_r \}$, onde

$t_r = (s_i , l_r , s_j)$ com $l_r = e_r/\lambda$, $e_r \in E|_s$, $e_r \notin E|_{si,sj}$

Operador 5: *evento trocado* (TraEveAlt)

Este operador troca cada evento associado a cada transição que tenha como origem um subestado do componente s pelos demais eventos de s que não estejam associados a essa transição.

Mutantes M_r , $0 \leq r \leq (k \,.\, (m^2 / 4))$, são gerados da seguinte maneira:

Para cada par de estados (s_i , s_j) onde $s_i \in \rho(s)$ e $s_j \in \rho(s) \vee s_j \in S$ tal que $T|_{si,sj} \neq \phi$ geram-se $(| E|_{si,sj} | \,.\, (m - | E|_{si,sj} |))$ mutantes onde:

$(E|_s)_r = E|_s$; $(A|_s)_r = A|_s$; $(\rho(s))_r = \rho(s)$; $(T|_s)_r \neq T|_s$;

$(T|_{si,sj})_r = T|_{si,sj} - \{ t \} \cup \{ t_r \}$,

$t = (s_i , l , s_j)$, $l = e/a$, e é composto pelos elementos de $E|_{si,sjt}$

$t_r = (s_i , l_r , s_j)$, $l_r = e_r /a$, e_r é composto pelos elementos de $E|_{si,sjt}$
onde $e_i = e'$, $e' \in E|_s$, $e' \notin E|_{si,sj}$, para cada $e_i \in E|_{si,sjt}$

Operador 6: *destino trocado* (TraDesStaAlt)

Este operador troca o estado destino associado a cada transição que tem como origem um subestado do componente s , pelos demais subestados de s , se o destino da transição é um subestado de s ou pelos demais subestados do componente s' , se o destino da transição é um subestado de s'.

Mutantes M_r , $0 \leq r \leq (m.k) . (k - 1) + \sum_{j=1}^{nt_{ext}} (ns_{ext} - 1)$, são gerados da seguinte maneira:

Para cada par de estados (s_i , s_j) onde $s_i \in \rho(s)$ e $s_j \in \rho(s) \vee s_j \in S$ tal que $T|_{si,sj} \neq \phi$ e para cada $t \in T|_{si,sj}$ geram-se mutantes onde:

$(E|_s)_r = E|_s$; $(A|_s)_r = A|_s$; $(\rho(s))_r = \rho(s)$; $(T|_s)_r \neq T|_s$;

$(T|_{si,sj})_r = T|_{si,sj} - \{ t \} \cup \{ t_r \}$, onde

$t = (s_i , l , s_j)$ e
se $s_j \in \rho(s)$ então:
$t_r = (s_i , l , s_k)$ para cada $s_k \in \rho(s)$

se $s_j \notin \rho(s)$ e $s_j \in \rho(s')$, $s' \in S$ então:
$t_r = (s_i , l , s_k)$ para cada $s_k \in \rho(s')$

Operador 7: *ação faltando* (TraActDel)

Este operador remove a ação associada a cada transição do componente s. No caso da transição possuir uma ação composta, é retirada uma ação de cada vez.

Mutantes M_r, $0 \leq r \leq ((m.k).n + nt_{ext} . n)$ são gerados da seguinte maneira:

Para cada par de estados (s_i , s_j) onde $s_i \in \rho(s)$ e $s_j \in \rho(s) \vee s_j \in S$ tal que $T|_{si,sj} \neq \phi$ e para cada $t \in T|_{si,sj}$ geram-se $| A|_{si,sjt} |$ mutantes onde:

$(E|_s)_r = E|_s$; $(A|_s)_r \subseteq A|_s$; $(\rho(s))_r = \rho(s)$; $(T|_s)_r \neq T|_s$;

$(T|_{si, sj})_r = T|_{si, sj} - \{ t \} \cup \{ t_r \}$,

$t = (s_i , l , s_j)$ com $l = e/(A|_{si,sjt})$, $e \in E|_{si,sj}$

$t_r = (s_i , l_r , s_j)$ com $l_r = e/((A|_{si,sjt})_r)$ onde

$(A|_{si,sjt})_r = (A|_{si,sjr}) - \{ a_r \}$, $a_r \in A|_{si,sjt}$

Operador 8: *ação trocada* (TraActAlt)

Este operador troca a ação associada a cada transição do componente s, pelas demais ações deste componente. No caso de ações compostas, a troca é realizada em cada uma das ações primitivas, desde que a ação que esteja sendo colocada não faça parte das demais ações associadas à mesma transição.

Mutantes M_r, $0 \leq r \leq (m.k) . (n - 1).$, são gerados da seguinte maneira:

Para cada par de estados (s_i , s_j) onde $s_i \in \rho(s)$ e $s_j \in \rho(s) \vee s_j \in S$ tal que $T|_{si,sj} \neq \phi$ e para cada $t \in T|_{si,sj}$ geram-se $(| A|_{si,sjlt} | . (n - | A|_{si,sjlt} |))$ mutantes onde:

$(E|_s)_r = E|_s$; $(A|_s)_r \subseteq A|_s$; $(\rho(s))_r = \rho(s)$; $(T|_s)_r \neq T|_s$;

$(T|_{si,sj})_r = T|_{si,sj} - \{ t \} \cup \{ t_r \}$,

$t = (s_i , l , s_j)$ com $l = e/(A|_{si,sjlt})$, $e \in E|_{si,sj}$

$t_r = (s_i , l_r , s_j)$ com $l_r = e/((A|_{si,sjlt})_r)$ onde

$(A|_{si,sjlt})_r = (A|_{si,sjlt}) - \{ a_q \} \cup \{ a_p \}$,

$a_q \in A|_{si,sjlt}$, $a_p \in A|_s$ e $a_p \notin A|_{si,sjlt}$

Operador 9: *estado faltando* (StaDel)

Este operador faz uma junção de dois subestados do componente s em um único, desde que exista uma transição conectando os mesmos. O estado resultante dessa junção passa a ser a origem e/ou destino de todas as transições que tinham como origem e/ou destino os estados que foram agrupados.

Mutantes M_r, $0 \leq r \leq C_2^k$, são gerados da seguinte maneira:

Para cada par de estados (s_i , s_j) onde $s_i \in \rho(s)$ e $s_j \in \rho(s) \vee s_j \in S$ tal que $T|_{si,sj} \neq \phi$ geram-se mutantes onde:

$(E|_s)_r = E|_s$; $(A|_s)_r = A|_s$; $(\rho(s))_r \neq \rho(s)$; $(T|_s)_r \neq T|_s$;

$(\rho(s))_r = \rho(s) - \{ s_i , s_j \} \cup \{ s' \}$, onde s' é a junção de s_i com s_j

$(T|_{sk,s'})_r = T|_{sk,si} \cup T|_{sk,sj}$

$(T|_{s',sk})_r = T|_{si,sk} \cup T|_{sj,sk}$

$(T|_{s',s'})_r = T|_{si,si} \cup T|_{sj,sj}$

Para todo $s_k \in \rho(s) \vee s_k \in S$ tal que: $T|_{si,sk} \neq \phi$

$T|_{sj,sk} \neq \phi$

$T|_{sk,si} \neq \phi$

$T|_{sk,sj} \neq \phi$

6.4.2 Operadores de Mutação para Statecharts que Abordam Aspectos de Máquinas de Estados Finitos Estendidas

Nesta seção definem-se os operadores de mutação para a técnica Statecharts que abordam aspectos relacionados às Máquinas de Estados Finitos Estendidas. São operadores que tratam aspectos relativos ao uso de variáveis e condições. No Apêndice A, item A.3, pode-se observar, através da LES restrita a MEFs Estendidas, as características que diferenciam esta técnica das técnicas Máquinas de Estados Finitos e Statecharts.

Para definirem-se os operadores de mutação que abordam os aspectos de Máquinas de Estados Finitos Estendidas, foram considerados os operadores de mutação definidos para a linguagem de programação C (Agrawal et al, 1989), uma vez que entre eles existem operadores que tratam do uso de variáveis e condições, e também os operadores de mutação propostos por Weyuker et al (1994), para expressões booleanas.

Quanto aos operadores apresentados em (Agrawal et al, 1989), as mutações são classificadas em: mutações de comandos, de operadores, de variáveis e de constantes. A cada uma dessas classes pertencem vários tipos de

operadores, sendo que apenas alguns deles aplicam-se no contexto de MEFs Estendidas.

Dos operadores definidos por Weyuker et al (1994), num total de cinco, alguns são coincidentes àqueles apresentados em (Agrawal et al, 1989), e todos eles foram considerados no conjunto de operadores aqui proposto. Esses operadores foram definidos, pois os autores utilizaram o conceito básico da Análise de Mutantes para determinar a capacidade em revelar erros de uma estratégia de teste proposta por eles, denominada *estratégia de impacto significativo*, que apóia a seleção de casos de teste para definições baseadas em expressões booleanas. A idéia dessa estratégia é que, dada uma fórmula booleana, devem ser selecionados, quando possível, casos de teste que sejam capazes de demonstrar que cada um dos literais da fórmula tem impacto significativo no seu resultado. Isso acontece quando considerando-se um determinado literal e um determinado caso de teste, se o valor desse literal for alterado, obtém-se um valor diferente para a fórmula. Assim, se um conjunto de casos de teste não é capaz de mostrar o impacto significativo para cada literal, então um literal pode ser erroneamente negado sem causar efeito no valor da fórmula. Nesse caso, ter-se-ia um erro não revelado. Os operadores definidos em (Weyuker et al, 1994) foram: "*variable negation fault*" - onde cada variável é substituída, uma de cada vez, pelo seu complemento; "*expression negation fault*" - que troca cada expressão pelo seu complemento; "*variable reference fault*" - que troca cada ocorrência de uma variável pelas outras variáveis e por constantes; "*operator reference fault*" - que troca um operador booleano por outro operador booleano e "*associative shift fault*" - que troca a associatividade dos termos.

Os operadores aqui definidos foram agrupados em quatro classes: mutações de expressões, de operadores, de variáveis e de constantes, de forma similar à classificação adotada em (Agrawal et al, 1989). Os operadores propostos por Weyuker et al (1994) foram inseridos nas classes de erros apropriadas. Em seguida, na Tabela 6.1, apresentam-se, as classes de mutações com seus respectivos operadores.

Tabela 6.1 - Síntese dos Operadores de Mutação para MEFs Estendidas

Classe de Mutação	*Operador*
Expressões	- exclusão de expressão - negação de expressão booleana - troca associatividade dos termos
Operadores	- troca operador aritmético por operador aritmético - troca operador relacional por operador relacional - troca operador lógico por operador lógico - negação lógica
Variáveis	- troca variável por variável - troca variável por constante
Constantes	- troca o valor de constantes por outros valores - troca constante por variável escalar

De acordo com a notação e definições apresentadas na Seção 6.2.1, o conjunto de operadores de mutação para MEFs Estendidas é definido com base nos conjuntos de transição, T_{est} , de expressões, V_{est} , de condições, C_{est} , de eventos E_{est} e de ações, A_{est} , os quais correspondem a restrições nos conjuntos T, V, C e E, respectivamente, apresentados naquela seção.

T_{est} = { t ∈ T | l = e/a, onde e e a estão definidos sobre os conjuntos V_{est} , C_{est} , E_{est} , A_{est} } a seguir:

V_{est} = V = { v ∈ V | v é definido como:

1. Se *k* é um número então $k \in V_{est}$,
2. Se $v \in V_p$ então $v \in V_{est}$,
3. Se $v \in V_{est}$ então *current (v)* $\in V_{est}$,
4. Se v_1, $v_2 \in V_{est}$ e *op* é uma operação algébrica então *op* $(v_1, v_2) \in V_{est}$ }

C_{est} = { c ∈ C | c é definido como:

1. *T, F* ∈ *C* ; *T, F* correspondem a *true* e *false* , respectivamente.
2. Se $c \in C_p$ então $c \in C_{est}$,

3. Se $u, v \in V$, $R \in \{=, >, <, \neq, \geq, \leq\}$ então $u\,R\,v \in C_{est}$,
4. Se $c \in C_{est}$ então *current(c)* $\in C_{est}$,
5. Se $c_1, c_2 \in C_{est}$ então $c_1 \vee c_2$, $c_1 \wedge c_2$, $\neg c_1 \in C_{est}$ }

E_{est} = { e ∈ E | e é definido como:

1. $\lambda \in E$; λ é o *evento nulo* ,
2. Se $e \in E_p$ então $e \in E_{est}$,
3. Se $c \in C$ então *true(c)* , *false(c)* $\in E_{est}$,
4. Se $v \in V$ então *changed(v)* $\in E_{est}$,
5. Se $e_1, e_2 \in E_{est}$ então $e_1 \vee e_2$, $e_1 \wedge e_2 \in E_{est}$,
6. Se $e \in E_{est}$, $c \in C$ então $e[c] \in E_{est}$ }

A_{est} = A = { a ∈ A | a é definido como:

1. $\mu \in A$, μ é a *ação nula* ,
2. Se $c \in C_p$, $d \in C$ então $c := d \in A_{est}$,
3. Se $v \in V_p$, $u \in V$ então $v := u \in A_{est}$,
4. Se $a_i \in A_{est}$, i = 0, ..., n então a_0 ; ... ; $a_n \in A_{est}$ }

Em seguida, apresentam-se, para cada classe de mutação, as definições dos operadores que compõem a classe:

- **Mutações de Expressões**

Esta classe de operadores modela mutação em expressões e/ou condições que podem ocorrer nos eventos e/ou ações das transições.

Operador 1: *exclusão de expressão* (TraExpDel)

Para cada $t \in T_{est}$, com $l = e/a$, em cada ocorrência de uma expressão ou de uma condição, exclui-se a expressão ou a condição.

ex.: i) $l = true(x + y > z)/a \Rightarrow l = \lambda/a$

ii) $l = e/v := x + y \Rightarrow l = e$

Operador 2: ***negação de expressão booleana*** **(TraNegBoolExp)**

Para cada $t \in T_{est}$, com $l = e/a$, em cada ocorrência de uma expressão booleana substitui-se a expressão booleana pela negação da mesma.

ex.: i) $l = \text{true}(x > y)/a \Rightarrow l = \text{true}(\text{not}(x > y))/a$

ii) $l = e/v := x > y \Rightarrow l = e/v := \text{not}(x > y)$

Operador 3: ***troca associatividade dos termos*** **(TraAssAlt)**

Para cada $t \in T_{est}$, com $l = e/a$, em cada ocorrência de expressão onde existe uma regra de associatividade, troca-se a associatividade.

ex.: i) $l = \text{true}(x \text{ and } (y \text{ or } z))/a \Rightarrow l = \text{true}((x \text{ and } y) \text{ or } z)/a$

ii) $l = e/v := x . (y + z) \Rightarrow l = e/ v := (x . y) + z$

- **Mutações de Operadores**

Esta classe de mutação modela a escolha de um operador incorreto em uma expressão.

Operador 4: ***troca operador aritmético por operador aritmético*** **(TraAritOperArit)**

Para cada $t \in T_{est}$, com $l = e/a$, em cada ocorrência de um operador aritmético em uma expressão, substitui-se o operador aritmético por outro operador aritmético.

ex.: i) $l = \text{true}(x + y > z)/a \Rightarrow l = \text{true}(x - y > z)/a$

ii) $l = e/v := x . y \Rightarrow l = e/ v := x + y$

Operador 5: ***troca operador relacional por operador relacional*** **(TraRelOperRel)**

Para cada $t \in T_{est}$, com $l = e/a$, em cada ocorrência de um operador relacional, substitui-se o operador relacional por outro operador relacional.

ex.: i) $l = \text{true}(x > y)/a \Rightarrow l = \text{true}(x < y)/a$

ii) $l = e/v := x > y \Rightarrow l = e/v := x < y$

Operador 6: ***troca operador lógico por operador lógico*** **(TraLogOperLog)**

Para cada $t \in T_{est}$, com $l = e/a$, em cada ocorrência de um operador lógico, substitui-se o operador lógico por outro operador lógico.

ex.: i) $l = e_1 \vee e_2/a \Rightarrow l = e_1 \wedge e_2/a$

ii) $l = e/v := (x > y) \wedge (x > z) \Rightarrow l = e/v := (x > y) \vee (x > z)$

Operador 7: ***negação lógica*** **(TraNegLogExp)**

Para cada $t \in T_{est}$, com $l = e/a$, em cada ocorrência de uma variável lógica substitui-se a variável lógica por sua negação.

ex.: i) $l = \text{true}(x \text{ or } y)/a \Rightarrow l = \text{true}(x \text{ or not } y)/a$

ii) $l = e/v := x \Rightarrow l = e/v := \text{not } x$

- **Mutações de Variáveis**

Esta classe de mutação modela o uso incorreto de identificadores nas expressões.

Operador 8: ***troca variável por variável*** **(TraVarAltVar)**

Para cada $t \in T_{est}$, com $l = e/a$, em cada ocorrência de uma expressão troca-se cada variável da expressão pelas demais variáveis utilizadas no Statecharts, uma de cada vez, respeitando-se o tipo da variável.

ex.: i) l = true(x > y)/a ⇒ l = true(x > z)/a

ii) l = e/v := x - y ⇒ l = e/v := x - z

Operador 9: *troca variável por constante* (TraVarAltCons)

Para cada $t \in T_{est}$, com l = e/a , em cada ocorrência de uma expressão troca-se cada variável da expressão por um valor constante utilizado no Statecharts, uma de cada vez.

ex.: i) l = true(x > y)/a ⇒ l = true(x > 3)/a

ii) l = e/v := x - y ⇒ l = e/v := x - 3

- **Mutações de Constantes**

Esta classe de mutação modela o uso de valores incorretos nas constantes utilizadas nas expressões.

Operador 10: *troca de constantes por constantes requeridas* (TraConsAltCons)

Para cada $t \in T_{est}$, com l = e/a , em cada ocorrência de uma expressão que faça uso de uma constante, aplica-se a seguinte alteração: troca-se o valor da constante por { 0, 1, -1 } para inteiros ou por { 0.0, 1.0, -1.0 } para reais.

ex.: i) l = true(x > 3)/a ⇒ l = true(x > -1)/a

ii) l = e/v := 3 ⇒ l = e/v := - 1

Operador 11: *troca constante por variável escalar* (TraConsAltVar)

Para cada $t \in T_{est}$, com l = e/a , em cada ocorrência de uma expressão que faça uso de uma constante, troca-se a constante pelas variáveis escalares utilizadas no Statecharts, uma de cada vez.

ex.: i) l = true(x > 3)/a ⇒ l = true(x > y)/a

ii) l = e/v := 3 ⇒ l = e/v := x

6.4.3 Operadores de Mutação para Aspectos Específicos da Técnica Statecharts

Nesta seção definem-se alguns operadores de mutação relativos aos aspectos intrínsecos à técnica Statecharts. A aplicação desses operadores provoca a geração de mutantes que auxiliam detectar erros relativos aos aspectos de concorrência e sincronização do sistema descrito pelo Statecharts em teste, uma vez que eles exploram características de broadcasting e paralelismo. Outra característica também explorada por esses operadores é a capacidade de história, também disponível nessa técnica. A seguir, eles são apresentados de acordo com a característica correspondente.

- **História**

Os operadores que abordam esse aspecto consideram as duas alternativas possíveis: o caso em que o estado possui um símbolo de história associado, definindo alguns operadores de mutação que alteram as condições inicialmente retratadas por esse aspecto, e o caso em que o estado não possui um símbolo de história associado, definindo alguns operadores que associam o símbolo de história ao estado.

Retomando as definições feitas anteriormente:

1- um estado $s \in S$ possui o símbolo de história associado a ele se

$\gamma^{-1}(s) = h \quad \vee \quad \gamma^{-1}(s) = h^*, \quad h, h^* \in H$;

2- $\forall\ t \in T \Rightarrow t = (X, l, Y)$

considere os seguintes conjuntos:

$T_{s/H}(s) = \{ t \in T \mid s \in Y \ \wedge\ t$ entra em s por história $\}$

$T_{s/H}(s) = \{ t \in T \mid s \in Y \ \wedge\ t$ entra em s sem história $\}$

Com base nessas definições, apresentam-se as definições dos operadores 1 a 5, para o caso em que o estado possui o símbolo de história associado, e os operadores 6 e 7, para o caso em que o estado não possui o símbolo associado:

Operador 1: *desassocia história da transição* (HistTraDel)

Este operador desassocia do símbolo de história, as transições que entram num estado através desse símbolo, uma de cada vez. Faz-se isso para todos os estados que possuem o símbolo de história associado a ele.

Mutantes M_r, $0 \leq r \leq \sum_{i=1}^{|S|} |T_{c/H}(s_i)|$, são definidos da seguinte maneira:

Para cada $s \in S$, tal que $\gamma^{-1}(s) \neq \phi$ geram-se $|T_{c/H}(s)|$ mutantes onde:

$T_{c/H}(s)_r = T_{c/H}(s) - \{t\}$ e

$T_{s/H}(s)_r = T_{s/H}(s) \cup \{t\}$

Operador 2: *troca transição associada ao símbolo de história* (HistTraAltTra)

Este operador desassocia o símbolo de história de uma transição que entra num estado através desse símbolo e associa o símbolo a uma transição que entra no estado sem ser por história.

Mutantes M_r, $0 \leq r \leq \sum_{i=1}^{|S|} (|T_{c/H}(s_i)| . |T_{s/H}(s_i)|)$, são definidos da seguinte maneira:

Para cada $s \in S$, tal que $\gamma^{-1}(s) \neq \phi$ geram-se $|T_{c/H}(s)| . |T_{s/H}(s)|$ mutantes onde:

$T_{c/H}(s)_r = T_{c/H}(s) - \{t_j\} \cup \{t_k\}$ e

$T_{s/H}(s)_r = T_{s/H}(s) \cup \{t_j\} - \{t_k\}$, para cada $t_j \in T_{c/H}(s)$ e cada $t_k \in T_{s/H}(s)$

Operador 3: *exclui história do estado* (HistDelSta)

Este operador elimina o símbolo de história que está associado a um estado s ; sendo assim, todas as transições passam a entrar em s sem ser por história.

Mutantes M_r, $0 \leq r \leq |S|$, são definidos da seguinte maneira:

Para cada $s \in S$ tal que $\gamma^{-1}(s) \neq \phi$ gera-se um mutante onde $\gamma^{-1}(s) = \phi$.

Obs.: no caso de existir apenas uma transição que entre no estado por história, esse operador é equivalente ao operador 1 que desassocia a história da transição.

Operador 4: *troca h por h (HistHH*)**

Este operador troca o tipo de história associada a um estado, fazendo com que a história passe a atingir todos os subestados na hierarquia sempre que o estado possuir mais de um nível de decomposição.

Mutantes M_r, $0 \leq r \leq |S|$, são definidos da seguinte maneira:

Para cada $s \in S$ tal que $\gamma^{-1}(s) = h$ e $\exists\ s' \in \rho(s)$ com $\rho(s') \neq \phi$
faz-se $\gamma^{-1}(s) = h^*$

Operador 5: *troca h* por h* (HistH*H)

Este operador troca o tipo de história associada a um estado, fazendo com que a história seja aplicada apenas no nível do estado.

Mutantes M_r, $0 \leq r \leq |S|$, são definidos da seguinte maneira:

Para cada $s \in S$ tal que $\gamma^{-1}(s) = h^*$ faz-se $\gamma^{-1}(s) = h$

Operador 6: *associa h ao estado* (HisInsHSta)

Este operador associa o símbolo de história h a um estado que não possui história, fazendo com que as transições passem a entrar no estado por história, uma de cada vez, de modo que a história atinja apenas o nível do estado.

Mutantes M_r, $0 \leq r \leq |S| . |T|$, são definidos da seguinte maneira:

Para cada $s \in S$ tal que $\rho(s) \neq \phi \wedge \psi(s) = OR \wedge \gamma^{-1}(s) = \phi$
faz-se $\gamma^{-1}(s) = h$ e para cada $t \in T_{s/H}(s)$ faz-se

$T_{s/H}(s)_r = T_{s/H}(s) - \{t\}$ e

$T_{c/H}(s)_r = \{t\}$

Operador 7: *associa h* ao estado* (HisInsH*Sta)

Este operador associa o símbolo de história h* a um estado que não possui história, fazendo com que as transições passem a entrar no estado por história, uma de cada vez, de modo que a história atinja todos os níveis de hierarquia do estado.

Mutantes M_r, $0 \leq r \leq |S| . |T|$, são definidos da seguinte maneira:

Para cada $s \in S$ tal que $\gamma^{-1}(s) = \phi \wedge \psi(s) = OR \wedge \rho(s_i) \neq \phi$ para algum $s_i \in \rho(s)$
faz-se $\gamma^{-1}(s) = h^*$ e para cada $t \in T_{s/H}(s)$ faz-se

$T_{s/H}(s)_r = T_{s/H}(s) - \{t\}$ e

$T_{c/H}(s)_r = \{t\}$

- **Paralelismo**

Os operadores definidos exploram algumas condições e eventos utilizados quando se têm estados do tipo AND. Para cada um desses eventos e condições apresentam-se suas definições e os operadores propostos:

condição *in(s)* : corresponde ao fato de estar ativado o estado s num componente paralelo àquele no qual essa condição aparece. Para essa condição definem-se dois operadores com base na seguinte consideração:

Seja $T_{c\text{-}in} = \{ t \in T \mid l = e[c]/a$ e c contém a condição in(s), $s \in S \}$

Operador 8: *exclui condição in(s)* (TraCondInDel)

Este operador exclui a condição in(s) de uma transição, fazendo com que não seja mais exigido que o estado s esteja ativo para que a transição possa ser disparada. Se a condição c que contém in(s) for uma condição composta, então exclui-se a condição in(s) juntamente com o operador "associado" a esta condição, obedecendo-se a ordem de prioridade dos operadores.

Mutantes M_r, $0 \leq r \leq | T_{c\text{-}in} |$, são definidos da seguinte maneira:
Para cada $t \in T_{c\text{-}in}$ exclui-se de t a condição in(s)

Operador 9: *troca estado da condição in(s)* (TraCondInStaAlt)

Este operador altera o estado que deve estar ativo para que a transição possa ser disparada, alternando o estado s por todos os outros estados pertencentes ao mesmo componente que s pertence.

Seja $s_i \in S$ tal que $s \in \rho(s_i)$

Mutantes M_r, $0 \leq r \leq \sum_{i=1}^{|T_{c\text{-}in}|} (| \rho(s_i) | - 1)$, são definidos da seguinte maneira:

Para cada $t \in T_{c\text{-}in}$ troca-se s por todo estado x tal que $x \in \rho(s_i)$ e $x \neq s$

condição *not-yet(e)* : corresponde ao fato de ainda não ter ocorrido o evento e. Para essa condição definem-se dois operadores com base na seguinte consideração:

Seja $T_{c\text{-}ny} = \{ t \in T \mid l = e[c]/a$ e c contém a condição not-yet(e), $e \in E \}$

Operador 10: *exclui condição not-yet(e)* **(TraCondNYDel)**

Este operador exclui a condição ny(e) de uma transição fazendo com que não seja mais exigido que o evento e ainda não tenha ocorrido para que a transição possa ser disparada. Se a condição c que contém ny(e) for uma condição composta, então exclui-se a condição ny(e) juntamente com o operador "associado" a esta condição, obedecendo-se a ordem de prioridade dos operadores.

Mutantes M_r, $0 \leq r \leq |T_{c\text{-}ny}|$, são definidos da seguinte maneira:
Para cada $t \in T_{c\text{-}ny}$ exclui-se de t a condição not_yet(e)

Operador 11: *troca evento da condição not-yet(e)* **(TraCondNYEveAlt)**

Este operador altera o evento que não pode ter ocorrido para que a transição possa ser disparada, alternando o evento e por todos os outros eventos existentes no sistema.

Mutantes M_r, $0 \leq r \leq |T_{c\text{-}ny}| . (|E| - 1)$, são definidos da seguinte maneira:
Para cada $t \in T_{c\text{-}ny}$ troca-se e por todo elemento e' tal que $e' \in E$, $e' \neq e$

evento *exit(s)* : corresponde ao fato de ter sido desativado um estado s num componente paralelo àquele no qual esse evento ocorre. Para esse evento definem-se dois operadores com base na seguinte consideração:

Seja $T_{c\text{-}ex} = \{ t \in T \mid l = e/a$ e e contém o evento exit(s), $s \in S \}$

Operador 12: *exclui evento exit(s)* (TraEveExDel)

Este operador deixa de exigir que o estado s tenha sido desativado para que a transição seja disparada. Caso o evento e da transição não seja composto, então o evento exit(s) é trocado pelo evento nulo λ fazendo com que a transição seja disparada sempre que seu estado origem estiver ativo. Caso o evento e da transição seja composto, então o evento exit(s) é excluído de e.

Mutantes M_r, $0 \leq r \leq |T_{e\text{-}ex}|$, são definidos da seguinte maneira:
Para cada $t \in T_{e\text{-}ex}$ onde $l = e/a$ gera-se um mutante tal que

se e não é composto então $l = \lambda/a$

se e é composto então $l = e'/a$, onde:
e' = e exceto pela exclusão do evento exit(s) que é excluído de e juntamente com o operador "associado" a este evento, obedecendo-se a ordem de prioridade dos operadores

Obs.: a aplicação desse operador é equivalente à aplicação do operador *evento faltando*, definido anteriormente na Seção 6.4.1, diferenciando-se apenas pelas características que são exploradas no contexto em que são aplicados.

Operador 13: *troca estado do evento exit(s)* (TraEveExStaAlt)

Este operador altera o estado que deve ter sido desativado para que a transição possa ser disparada, alternando o estado s por todos os outros estados pertencentes ao mesmo componente que s pertence.

Seja $s_i \in S \mid s \in \rho(s_i)$
Mutantes M_r, $0 \leq r \leq \sum_{i=1}^{|T_{e\text{-}ex}|} (|\rho(s_i)| - 1)$, são definidos da seguinte maneira:

Para cada $t \in T_{e\text{-}ex}$ troca-se s por todo estado x tal que $x \in \rho(s_i)$ e $x \neq s$

evento *entered(s)* : corresponde ao fato do estado s ter sido ativado em um componente paralelo àquele no qual esse evento ocorre. Para esse evento definem-se dois operadores com base na seguinte consideração:

Seja $T_{e\text{-}en} = \{ t \in T \mid l = e/a$ e e contém o evento entered(s), $s \in S \}$

Operador 14: *exclui evento entered(s)* **(TraEveEnDel)**

Este operador deixa de exigir que o estado s tenha sido ativado para que a transição possa ser disparada. Caso o evento e da transição não seja composto, então o evento entered(s) é trocado pelo evento nulo λ fazendo com que a transição seja disparada sempre que seu estado origem estiver ativo. Caso o evento e da transição seja composto, então o evento entered(s) é excluído de e.

Mutantes M_r, $0 \leq r \leq | T_{e\text{-}en} |$, são definidos da seguinte maneira:
Para cada $t \in T_{e\text{-}en}$ gera-se um mutante tal que

se e não é composto então $l = \lambda/a$

se e é composto então $l = e'/a$, onde:
$e' = e$ exceto pela exclusão do evento entered(s) que é excluído de e juntamente com o operador "associado" a este evento, obedecendo-se a ordem de prioridade dos operadores.

Obs.: a aplicação desse operador é equivalente à aplicação do operador *evento faltando*, definido anteriormente na Seção 6.4.1, diferenciando-se apenas pelas características que são exploradas no contexto em que são aplicados.

Operador 15: *troca estado do evento entered(s)* (TraEveEnStaAlt)

Este operador altera o estado que deve ter sido ativado para que a transição possa ser disparada, alternando o estado s por todos os outros estados pertencentes ao mesmo componente que s pertence.

Seja $s_i \in S \mid s \in \rho(s_i)$

Mutantes M_r, $0 \leq r \leq |T_{e\text{-}en}| . (|\rho(s_i)| - 1)$, da seguinte maneira:

Para cada $t \in T_{e\text{-}en}$ troca-se s por todo estado x tal que $x \in \rho(s_i)$ e $x \neq s$

- **Broadcasting**

Os operadores definidos exploram os aspectos de sincronização do sistema através das ações geradas e executadas em componentes paralelos, em uma reação em cadeia. Antes de serem apresentados os operadores, sejam as seguintes considerações:

$\forall\ t \in T \Rightarrow t = (X, l, Y)$

$T_{br} = \{ t \in T \mid l = e/a$ onde a contém um evento interno, ou seja,

$\exists\ t' \in T, t' = (X', l', Y')$ e $l' = a/a' \}$

Para cada $t_i \in T_{br}$ sejam:

$O_{bri} = LCA^+(X_i)$ o estado OR no qual t_i é disparada originando o broadcasting (origem do broadcasting)

$D_{bri} = \{ x' \in S \mid x' = LCA^+(X'),$ com $X' = source(t') \}$

o conjunto de estados OR nos quais a ação gerada por t_i é executada, disparando as transições t' (destinos do broadcasting)

$T_{Obri} = \{ t \in T \mid source(t) \in \rho(O_{bri}) \}$

o conjunto das transições que possuem como estado origem os estados que pertencem ao componente OR onde o broadcasting é originado

$T_{Dbri} = \{ t' \in T \mid source(t') \in \rho(x') , x' \in D_{bri} \}$

o conjunto das transições que possuem como estado origem os estados que pertencem aos componentes OR que são os destinos do broadcasting

Definem-se então os operadores que exploram a característica de broadcasting da seguinte maneira:

Operador 16: *altera transição origem do broadcasting* (TraBrOrigAlt)

Este operador altera a transição que contém a ação que dá origem ao broadcasting, colocando essa ação em cada uma das demais transições existentes no mesmo componente do Statecharts.

Mutantes Mr , $0 \leq r \leq \sum_{i=1}^{|T_{br}|} (|T_{Obri}| - 1)$, são definidos da seguinte maneira:

Para cada $t_i \in T_{br}$ tem-se $l_i = e_i / a_1;...br_i;...a_n$

faz-se $l_i = e_i / a_1;..a_n$ e

$l_j = e_j / a_j ; br_i$ onde l_j é o rótulo de $t_j \in T_{Obri}$, $t_j \neq t_i$ para toda $t_j \in T_{Obri}$

Operador 17: *altera transição destino do broadcasting* (TraBrDesAlt)

Este operador altera a transição que recebe a ação gerada pelo broadcasting, colocando essa ação em cada uma das demais transições existentes no mesmo componente do Statecharts.

Mutantes Mr, $0 \leq r \leq \sum_{i=1}^{|T_{br}|} (|T_{Dbri}| - 1)$, são definidos da seguinte maneira:

Para cada $t_i \in T_{br}$ tem-se $l_i = e_i / a_1;...br_i;...a_n$
Existem $n \geq 1$ transições t' tal que $l' = e'/a'$ e e' contém br_i

Para cada um dos n componentes $x' \in D_{bri}$ geram-se mutantes onde
$l_j' = e_j' \vee br_i / a_j'$ onde l_j' é o rótulo de $t_j' \in T_{Dbri}$, $t_j' \neq t'$
para toda $t_j' \in T_{Dbri}$

6.4.4 Mutação Alternativa para o Teste de Statecharts

Como os resultados obtidos da aplicação dos critérios de mutação alternativa no contexto de Máquinas de Estados Finitos e de Redes de Petri mostraram evidências de que o custo de aplicação do critério Análise de Mutantes pode ser reduzido significativamente, sem comprometer sua eficácia, propõem-se, também para o teste de Statecharts, o Critério de Mutação Aleatória, no qual examina-se uma pequena porcentagem de mutantes, e o Critério de Mutação Restrita, no qual examinam-se apenas alguns tipos específicos de mutantes.

Ressalta-se que esses critérios podem ser combinados com as três estratégias propostas ne Seção 6.4, como é apresentado na Tabela 6.2:

Tabela 6.2 - Possibilidades de Combinação entre os Critérios de Mutação Alternativa e as Estratégias de Teste para Statecharts

Estratégias de Teste / *Critérios de Mutação Alternativa*	Básica	Baseada em História	Baseada em Ortogonalidade
Critério de Mutação Aleatória	x	x	x
Critério de Mutação Restrita	x	x	x

6.5 Considerações Finais

Apresentou-se neste capítulo a aplicação do critério Análise de Mutantes na validação de especificações baseadas em Statecharts. Em geral, essas especificações são testadas através de simulação, com os seguintes tipos de execuções: interativa, em batch, programada e exaustiva (Harel, 1992).

No entanto, através de simulações, nem todos os erros são detectados, e além disso, não se tem critério para avaliar a atividade de teste de forma quantitativa. Como foi mostrado na Seção 6.3, gerou-se uma seqüência de eventos pela execução programada, para uma especificação Statecharts e, em seguida, aplicou-se essa seqüência na própria especificação e em um mutante da mesma, utilizando-se a execução interativa. O resultado obtido foi que tanto a especificação original como a especificação do mutante apresentaram o mesmo comportamento, observado através de suas configurações. Com isso, obtiveram-se então evidências de que, com a aplicação do critério Análise de Mutantes, as seqüências de teste podem ser enriquecidas à medida que o testador é forçado a construir novas seqüências para matar os mutantes que permanecem vivos durante uma atividade de teste.

Além disso, a aplicação da Análise de Mutantes permite que o teste seja conduzido de forma incremental, através da seleção dos operadores de mutação, definindo-se um conjunto inicial e aumentando-se esse conjunto de acordo com os recursos disponíveis. Particularmente, no caso de Statecharts, essa forma incremental pode ser considerada também do ponto de vista hierárquico, isto é, selecionando-se para teste determinados componentes da especificação, utilizando-se as estratégias propostas na Seção 6.4 para abstrair os componentes desejados.

Foram propostas três estratégias para selecionar os componentes de uma especificação. A Estratégia Básica, que abstrai todos os componentes do tipo OR do Statecharts, por nível de hierarquia, os quais correspondem às MEFs Estendidas que compõem a especificação. A Estratégia Baseada em História, que abstrai os componentes que possuem essa característica, permitindo que esse aspecto seja explorado separadamente e a Estratégia Baseada em Ortogonalidade, que abstrai os estados do tipo AND, possibilitando que a comunicação através de broadcasting seja tratada com maior detalhe.

Além dessas estratégias, foram definidos operadores de mutação que enfocam também três contextos diferentes, isto é, características particulares de Máquinas de Estados Finitos, a utilização de variáveis e condições, que caracteriza as Máquinas de Estados Finitos Estendidas e operadores que exploram aspectos específicos da técnica Statecharts, relacionados com história, broadcasting e paralelismo. Além dos propostos, outros operadores de mutação podem ser definidos com a finalidade de explorar outras condições de erro, permitindo uma melhor adequação do teste aos erros mais freqüentemente cometidos pelo especificador. Ressalta-se que com a implementação desses operadores de mutação suas definições poderão ser efetivamente avaliadas e refinadas.

Certamente, a condução de um estudo empírico para a técnica Statecharts é uma tarefa prioritária como continuidade deste trabalho. Salienta-se, no entanto, que os resultados já obtidos, considerando somente Máquinas de Estados Finitos, são um indicador dos benefícios que podem ser alcançados pelo uso do critério Análise de Mutantes no contexto de Statecharts, uma vez que a técnica

Statecharts é uma extensão das Máquinas de Estados Finitos. Assim, os conceitos e definições fornecidos neste capítulo constituem subsídios básicos e fundamentais para a evolução deste trabalho.

Statecharts é uma extensão das Máquinas de Estados Finitos. Assim, os conceitos e definições fornecidos neste capítulo constituem subsídios básicos e fundamentais para a evolução deste trabalho.

CAPÍTULO 7

PROTEUM-RS: UMA FERRAMENTA PARA APOIAR A APLICAÇÃO DO CRITÉRIO ANÁLISE DE MUTANTES NO CONTEXTO DE SISTEMAS REATIVOS

7.1 Considerações Iniciais

O objetivo deste capítulo é apresentar os principais aspectos funcionais e operacionais da ferramenta Proteum-RS que apóia a aplicação do critério Análise de Mutantes na validação de especificações do aspecto comportamental de Sistemas Reativos. Sua implementação está baseada na ferramenta Proteum/C (PROgram TEsting Using Mutants - versão C), (Delamaro, 1993) – que apóia a aplicação desse mesmo critério no teste de programas em C – e também em alguns módulos do ambiente StatSim, que apóia a edição e simulação de Statecharts (Masiero et al, 1991).

Denomina-se o "corpo básico" da ferramenta como Proteum-RS, isto é, Proteum - Reactive Systems, que foi construído a partir da Proteum/C. Denomina-se Proteum-RS/FSM, Proteum-RS/ST e Proteum-RS/PN as instanciações da Proteum-RS para Máquinas de Estados Finitos, Statecharts e Redes de Petri, respectivamente. O capítulo ilustra a ferramenta Proteum-RS através da versão FSM, da qual se tem um protótipo implementado.

Ressalta-se a importância do desenvolvimento de ferramentas que apóiem a atividade de teste, pois isso permite aumentar a qualidade da atividade de teste através da sua automatização e, como conseqüência, a qualidade do produto sendo desenvolvido. Particularmente, em relação ao critério Análise de Mutantes, sua aplicação de forma manual, como foi feito para Máquinas de Estados Finitos e Redes de Petri, a título de experimento piloto, é inviável. O número de mutantes gerados é muito grande, o que consome muito tempo além dos vários erros que podem ocorrer durante a geração e execução dos mesmos.

O capítulo está organizado da seguinte maneira: na Seção 7.2 apresentam-se as principais características da ferramenta Proteum-RS/FSM. Na Seção 7.3 apresentam-se alguns aspectos de implementação. Na Seção 7.4, apresentam-se seus aspectos operacionais e na Seção 7.4, as considerações finais.

7.2 Características Básicas da Ferramenta Proteum-RS/FSM

Em geral, pode-se identificar três contextos distintos de uso de uma ferramenta de teste: ensino, pesquisa e indústria. Considerando o estágio de aplicação da Análise de Mutantes na validação do aspecto comportamental de Sistemas Reativos, dá-se ênfase neste trabalho aos contextos de ensino e pesquisa. Nessa direção, uma característica comum é o apoio à realização de estudos empíricos que comparem as diversas técnicas/critérios de teste e validação. Um aspecto fundamental é facilitar a definição/seleção de operadores de mutação em apoio à aplicação da Análise de Mutantes Alternativa, contribuindo para a avaliação do custo e eficácia desses operadores.

Segundo (Delamaro e Maldonado, 1996a) uma ferramenta de teste baseada em mutação deve fornecer um conjunto mínimo de operações:

- *manipulação de casos de teste*, possibilitando a execução, inclusão/exclusão e desabilitação/habilitação dos casos de teste;
- *manipulação de mutantes*, permitindo a criação, seleção, execução e análise dos mutantes;
- *análise de adequação*, permitindo calcular o escore de mutação e fornecer relatórios estatísticos.

A ferramenta Proteum-RS, como será ilustrado na próxima seção, satisfaz esse conjunto mínimo de requisitos. Ela apóia as atividades básicas para a aplicação do critério Análise de Mutantes, permitindo ao testador a execução das seguintes tarefas:

- definição de casos de teste;
- execução da especificação em teste;
- geração de mutantes;
- execução dos mutantes;
- análise dos mutantes vivos;
- cálculo do escore de mutação.

A seqüência dessas atividades é característica do critério Análise de Mutantes e portanto, é basicamente a mesma disponível na ferramenta Proteum/C (Delamaro, 1993). A diferença fundamental reside na definição de casos de teste e na geração de mutantes, pois a ferramenta Proteum-RS/FSM atende à validação de especificações de Sistemas Reativos baseadas em Máquinas de Estados Finitos. Assim, a estrutura básica da Proteum/C foi reutilizada fazendo-se as adequações necessárias para possibilitar o teste de Máquinas de Estados Finitos.

Em seguida, apresentam-se os passos do critério Análise de Mutantes apoiados pela ferramenta Proteum-RS, exemplificados através da versão FSM que já está implementada. O Diagrama de Fluxo de Dados da Figura 7.1 apresenta uma visão funcional da ferramenta.

O processo "Editar Especificação da MEF" corresponde à edição da Máquina de Estados Finitos gerando-se uma descrição da mesma através da LES. Essa tarefa deve ser realizada utilizando-se um editor de texto disponível no sistema, através do qual cria-se o arquivo com a especificação da MEF. Esse arquivo é então utilizado na ferramenta, para a realização dos testes.

O processo "Abrir Teste" se encarrega de iniciar uma nova sessão de teste ou abrir uma sessão que já havia sido iniciada anteriormente. Isso é feito com base em um arquivo que armazena as informações sobre as sessões de teste. Observa-se que a ferramenta possibilita a realização do teste em sessões, permitindo que uma sessão seja interrompida e retomada posteriormente.

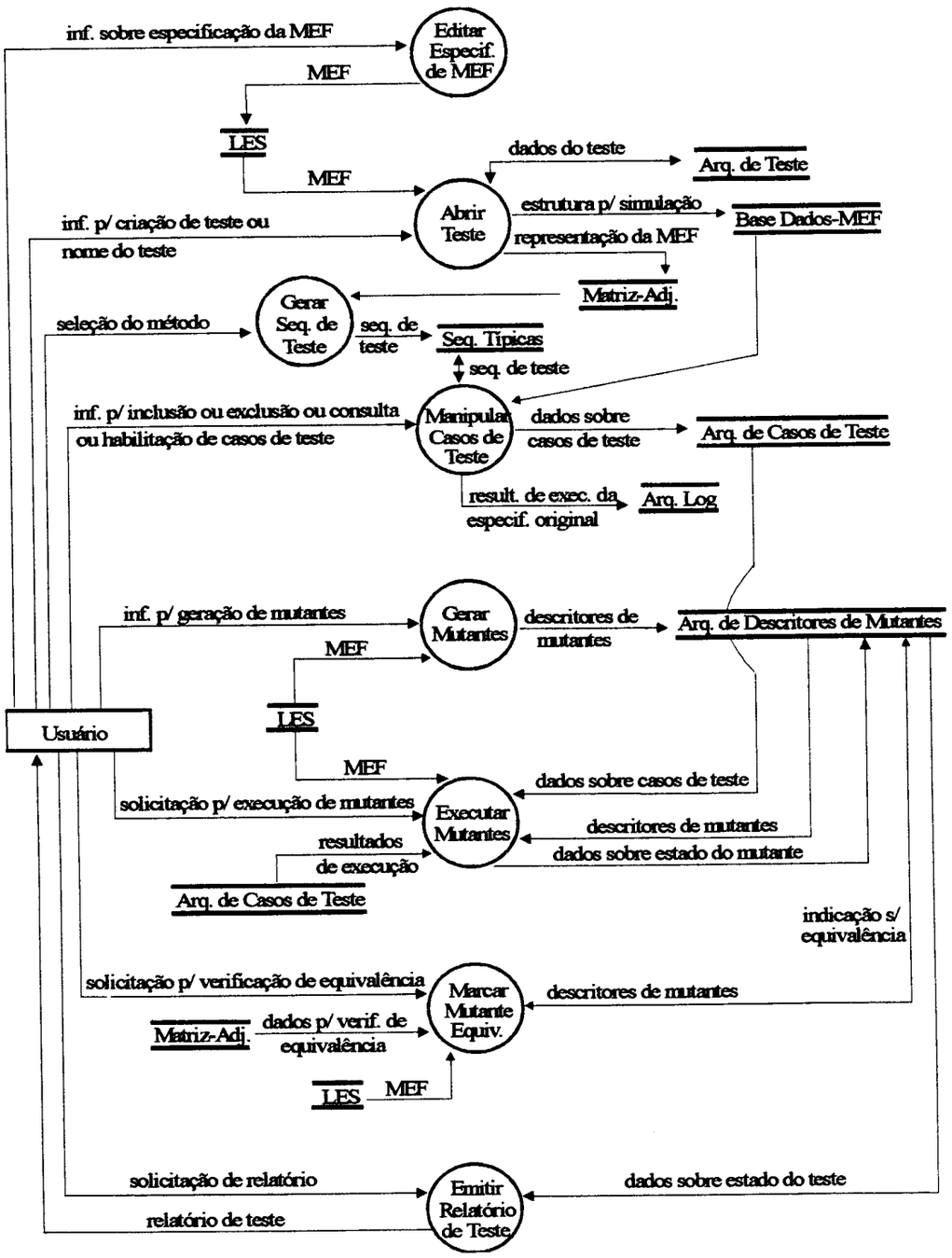


Figura 7.1 - Diagrama de Fluxo de Dados (DFD) da Ferramenta Proteum-RS/FSM

O processo "Manipular Casos de Teste" representa todas as operações que podem ser realizadas nesse sentido, ou seja, inserir, eliminar, habilitar, desabilitar e importar casos de teste. Para a inserção de casos de teste é necessário que a MEF seja simulada, através de uma seqüência de eventos a fim de que o seu comportamento seja avaliado pelo testador. Assim, torna-se fundamental o uso de um simulador para realizar essa tarefa, permitindo que o usuário defina uma seqüência de eventos que caracterize um caso de teste. Como os Statecharts são uma extensão das Máquinas de Estados Finitos e o ambiente StatSim (Masiero et al, 1991) apóia a edição e simulação de Statecharts, decidiu-se reutilizar os módulos desse ambiente necessários à simulação da MEF.

O processo "Gerar Seqüências de Teste" está representando, no DFD, a ferramenta MGASET (Nakazato, 1995), que fornece as propriedades da MEF e gera as seqüências de teste pelos métodos clássicos da literatura (Sabnani e Dahbura, 1988; Gonenc, 1970; Naito e Tsunoyama, 1981; Chow, 1978; Fujiwara et al, 1991). Esse processo ainda não está acoplado à ferramenta, mas seu acoplamento facilitará a realização de estudos empíricos para avaliar a adequação das seqüências geradas por esses métodos, em relação à Análise de Mutantes.

O processo "Gerar Mutantes" representa a geração dos mutantes da MEF que está sendo testada, com base no conjunto de operadores de mutação que foi definido no Capítulo 4. Essa geração pode ser restrita escolhendo-se apenas alguns tipos de operadores a serem aplicados, como pode também ser aleatória, escolhendo-se apenas uma porcentagem de mutantes a serem gerados de cada um dos tipos de operadores escolhidos.

O processo "Executar Mutantes" representa a simulação de todos os mutantes gerados com os casos de teste que estão habilitados no momento. A execução dos mutantes diferencia parcialmente da versão C, pois novamente são MEFs mutantes que devem ser simuladas e não programas mutantes na linguagem C. No entanto, todas as informações produzidas após a execução dos mutantes e que devem ser armazenadas, são as mesmas, isto é, o estado do mutante (vivo ou morto), o caso de teste que o matou, etc. Realizada a simulação do mutante, seu comportamento é comparado ao comportamento da MEF original. Para tal comparação utilizam-se as seqüências de saídas produzidas, com base

nas seqüências de entradas correspondentes aos casos de teste usados na simulação. Feita a comparação, o mutante é marcado como morto, caso as seqüências de saída sejam distintas, ou como vivo, caso contrário.

O processo "Marcar Mutante Equivalente" é executado com a intervenção do testador, que deve analisar os mutantes que permaneceram vivos a fim de decidir sobre sua equivalência com a MEF original. Como foi comentado no Capítulo 4, para uma determinada classe de máquinas, é possível decidir-se sobre a questão de equivalência automaticamente. Existe um algoritmo, proposto por Kohavi (1978), que será implementado e acoplado à Proteum-RS/FSM, de forma que para essa classe de MEFs, essa questão possa ser decidida de maneira automatizada.

O processo "Emitir Relatório de Teste" refere-se ao fornecimento de informações sobre a sessão de teste que está sendo conduzida, como por exemplo, o número de mutantes gerados, o número de mutantes mortos, o escore de mutação, etc.

Mantendo a característica de multilinguagem da ferramenta Proteum/C, a ferramenta Proteum-RS pode ser instanciada para outras técnicas de especificação.

A instanciação da Proteum-RS para a técnica Statecharts corresponde basicamente à implementação dos operadores de mutação e das estratégias definidas para esta técnica, pois a estrutura da versão ST é a mesma da versão FSM. Na versão FSM os módulos do ambiente StatSim são utilizados de forma restrita sendo que essa restrição é transparente a esses módulos, isto é, eles não sofreram alteração e continuam permitindo a simulação de Statecharts. Na verdade, a restrição às Máquinas de Estados Finitos corresponde essencialmente à forma de utilização da LES. Em outras palavras, para especificar uma MEF necessita-se de um subconjunto da LES, dado que uma MEF pode ser vista como um Statecharts formado de um único componente, assim como se pode dizer que cada componente básico de um Statecharts é uma MEF.

A instanciação da Proteum-RS para apoiar o teste de Redes de Petri já não é tão imediata, como a passagem da versão FSM para a versão ST, pois no caso de Redes de Petri torna-se necessário o uso de um simulador para essa técnica. Tendo-se disponível esse simulador, a instanciação da Proteum-RS para gerar-se a versão Proteum-RS/PN corresponde à implementação dos operadores de mutação definidos neste trabalho, tendo-se que fazer uma adequação à linguagem de especificação de Redes de Petri considerada no contexto do simulador.

7.3 Aspectos de Implementação

Em uma ferramenta de apoio ao teste de mutação o aspecto mais crítico está relacionado à geração e execução dos mutantes. Sabe-se que o número de mutantes gerados é, normalmente, bastante grande o que eleva o custo de aplicação do critério. Assim, com o objetivo de reduzir esse problema, uma maneira de gerar os mutantes é através da criação de descritores de mutação. Esses descritores são construídos a partir da LES da MEF original e dos operadores de mutação selecionados. Para cada um dos operadores, identificam-se os pontos da LES original que devem ser alterados para gerar cada um dos mutantes. Considere o exemplo da Figura 7.2.

LES da MEF original	LES da MEF mutante
estado protocolo ou	estado protocolo ou
subestados estado1 def, estado2, estado3, estado4	subestados estado1 def, estado2, estado3, estado4
estado estado1 atomo	estado estado1 atomo
estado estado2 atomo	estado estado2 atomo
estado estado3 atomo	estado estado3 atomo
estado estado4 atomo ;	estado estado4 atomo ;
evento cc origem estado1 destino estado1	evento cc origem estado1 destino estado1
evento cc origem estado2 destino estado1 acao {ERR} ;	evento cc origem estado2 destino estado1 ;
.....	.....

Figura 7.2 - Exemplo de um Mutante Através da LES

De acordo com esse exemplo, o mutante da LES original é gerado a partir de um descritor de mutação que contém as seguintes informações: a posição inicial a partir da qual a mutação será aplicada – posição 138ª do texto; o tamanho da cadeia de caracteres que será substituída na LES original – 10 (dez) caracteres correspondentes à cadeia "acao {ERR}" e a cadeia de caracteres que será inserida no lugar da cadeia de caracteres removida – correspondente a brancos. Observa-se que, dependendo da mutação a ser realizada na LES original, um mutante pode ter associado a ele, um ou mais descritores de mutação, quando mais do que uma cadeia tem que ser retirada/substituída.

Com essa solução, os mutantes só são gerados quando realmente necessários. Ou seja, na execução dos mutantes, onde eles são gerados um de cada vez e descartados após sua execução e na visualização dos mutantes quando o testador desejar analisá-los. Assim, obtém-se as seguintes vantagens: primeiro, economia de espaço, pois os mutantes permanecem na memória apenas temporariamente, nos momentos em que são utilizados e segundo, que se obtém maior rapidez na tarefa de simulação dos mutantes, pois a LES original já foi analisada anteriormente e os pontos de alteração estão armazenados no descritor de mutação.

A ferramenta está implementada na linguagem de programação C, roda em estações de trabalho sob o ambiente OpenWindows 3.0 e utiliza rotinas do X-View para a construção da interface com o usuário.

7.4 Aspectos Operacionais

Em seguida, os aspectos operacionais da Proteum-RS/FSM são exemplificados utilizando-se as janelas que compõem a interface com o usuário. Como mostra a Figura 7.3, com a janela principal da ferramenta, as opções disponíveis permitem a realização dos processos mencionados na seção anterior.

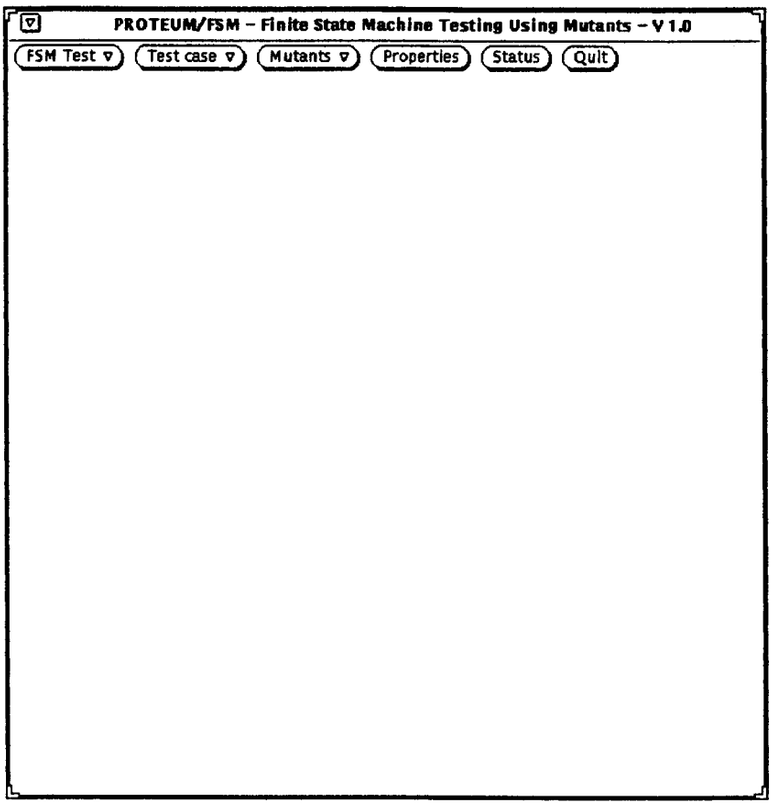


Figura 7.3 - Tela Principal da Ferramenta Proteum-RS/FSM

Através da opção *New* do botão *FSM Test* do Menu Principal uma nova Sessão de Teste é criada; em resposta à seleção de tal opção, abre-se uma janela - Figura 7.4 - onde são fornecidos os dados necessários para a criação de uma nova sessão de teste, como o diretório onde está armazenada a descrição da MEF que será testada e o nome que se deseja dar ao teste que está sendo criado. Ao se confirmar tais dados selecionando-se o botão *Confirm*, será chamado o módulo Analisador do ambiente Statsim, o qual, além de analisar a LES fornecida, irá gerar as estruturas de dados que serão usadas na implementação do algoritmo de equivalência de MEF (Gill, 1962).

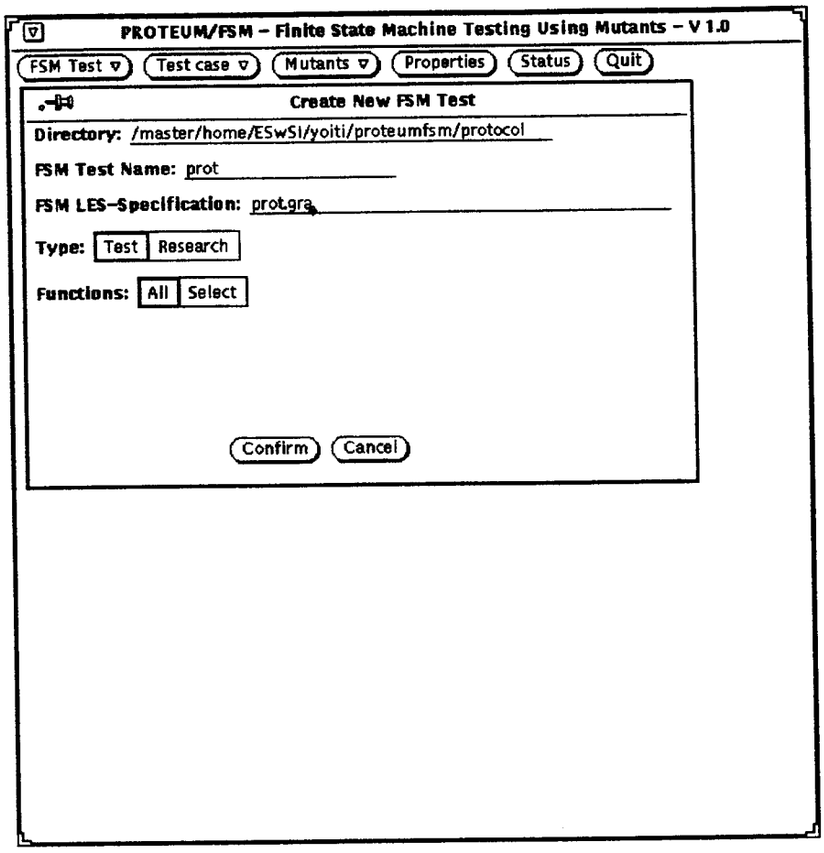


Figura 7.4 - Criação de uma Nova Sessão de Teste

A opção *Generate* do botão *Mutants* do Menu Principal abre uma janela com a relação das classes de operadores de mutação para MEFs, onde os operadores foram agrupados de acordo com a estrutura da MEF que é alterada - Figura 7.5. Nesse instante, pode-se escolher a porcentagem de mutantes que se deseja gerar de cada uma dessas classes e, selecionando-se os botões correspondentes a tais classes, é possível ser ainda mais criterioso no tipo de erro que se deseja explorar, escolhendo-se porcentagens individuais para cada operador. Ao selecionar-se o botão *Confirm*, é dado início à geração dos mutantes. Essas facilidades viabilizam o estabelecimento e o estudo de métodos/critérios alternativos da Análise de Mutantes para MEF, a exemplo dos estudos conduzidos por Mathur e Wong (1993).

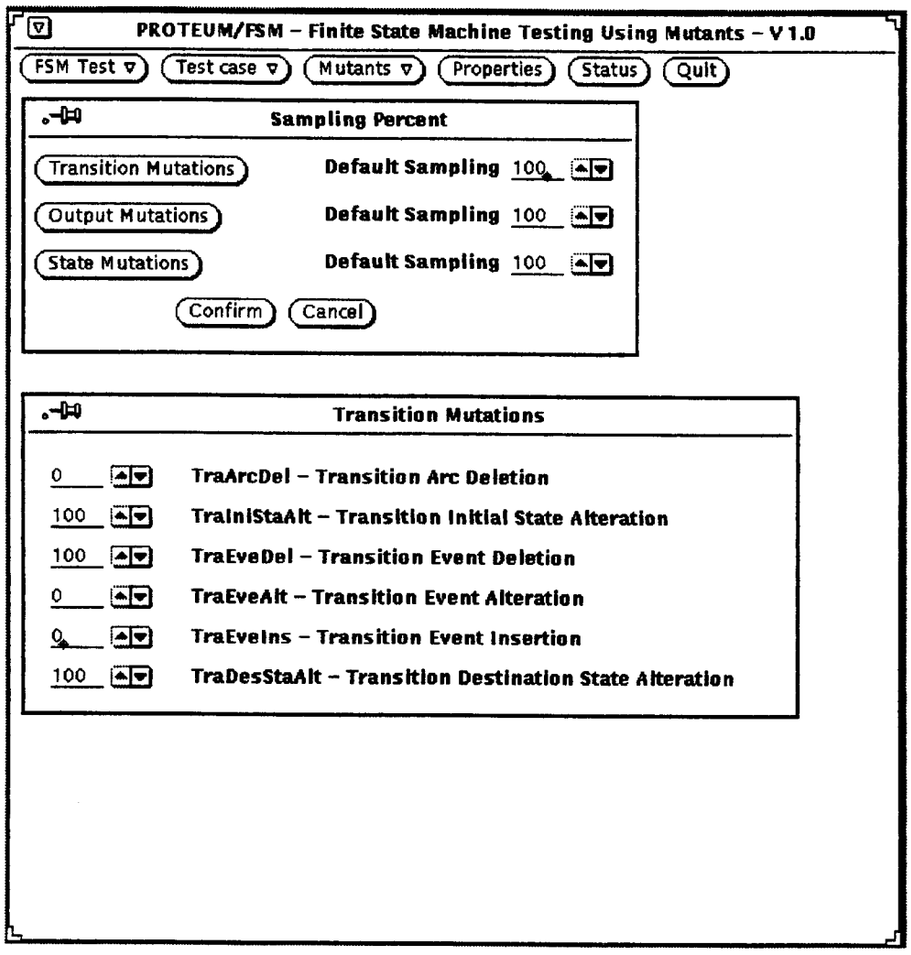


Figura 7.5 - Geração de Mutantes/Escolha dos Operadores de Mutação

Depois de gerados os mutantes, para que eles possam ser executados, deve-se criar casos de teste, isto é, seqüências de entradas que testem o comportamento da MEF. Isso é feito através do botão *Test Case* do Menu Principal. A opção *Add* permite que casos de teste sejam adicionados na Base de Dados através da simulação da MEF original, de forma interativa, onde o usuário fornece uma seqüência de teste - Figura 7.6. Nesse ponto, cabe ao testador analisar o resultado dessa simulação e confirmar a inclusão do caso de teste, se o

comportamento da MEF estiver de acordo com o esperado ou interromper o teste em caso contrário, pois um erro foi revelado. A opção *View* permite que se vejam os casos de teste armazenados na Base de Dados, Figura 7.7. A opção *Delete* permite que os casos de teste sejam removidos da Base de Dados. A opção *Generate* permite que sejam geradas seqüências de teste segundo os métodos utilizados para validação de MEFs, e corresponde ao acoplamento que será feito do módulo MGASET (Nakazato, 1995) à ferramenta Proteum-RS/FSM. A opção *Import* permite que casos de teste sejam importados de outras ferramentas; essa opção não está ainda operacional.

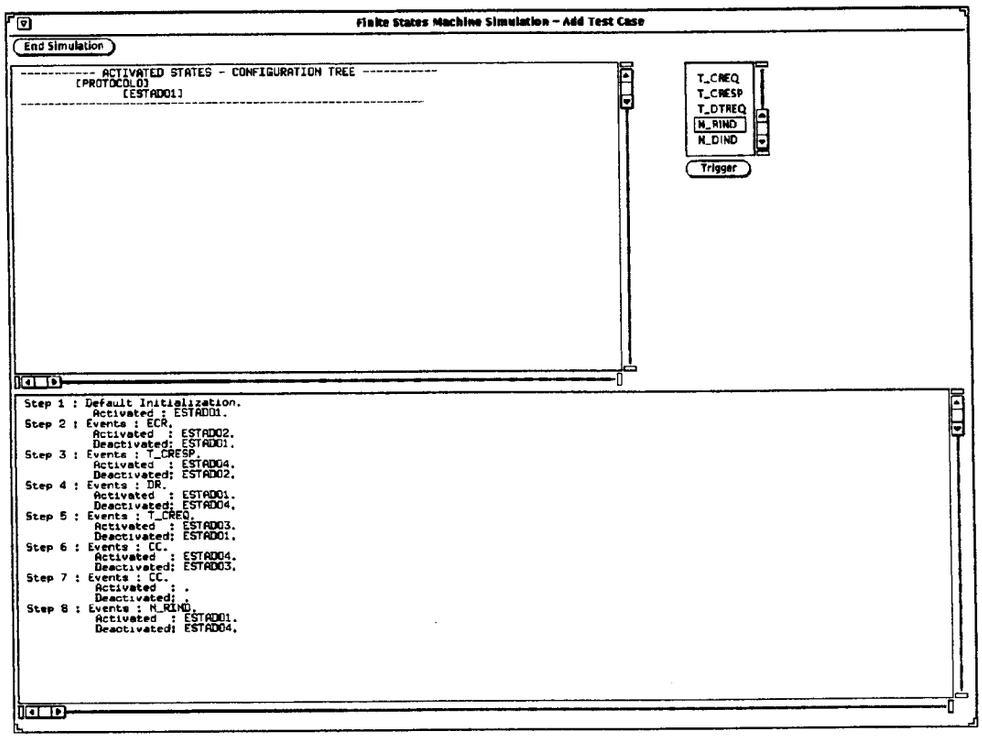


Figura 7.6 - Adição de um Novo Caso de Teste

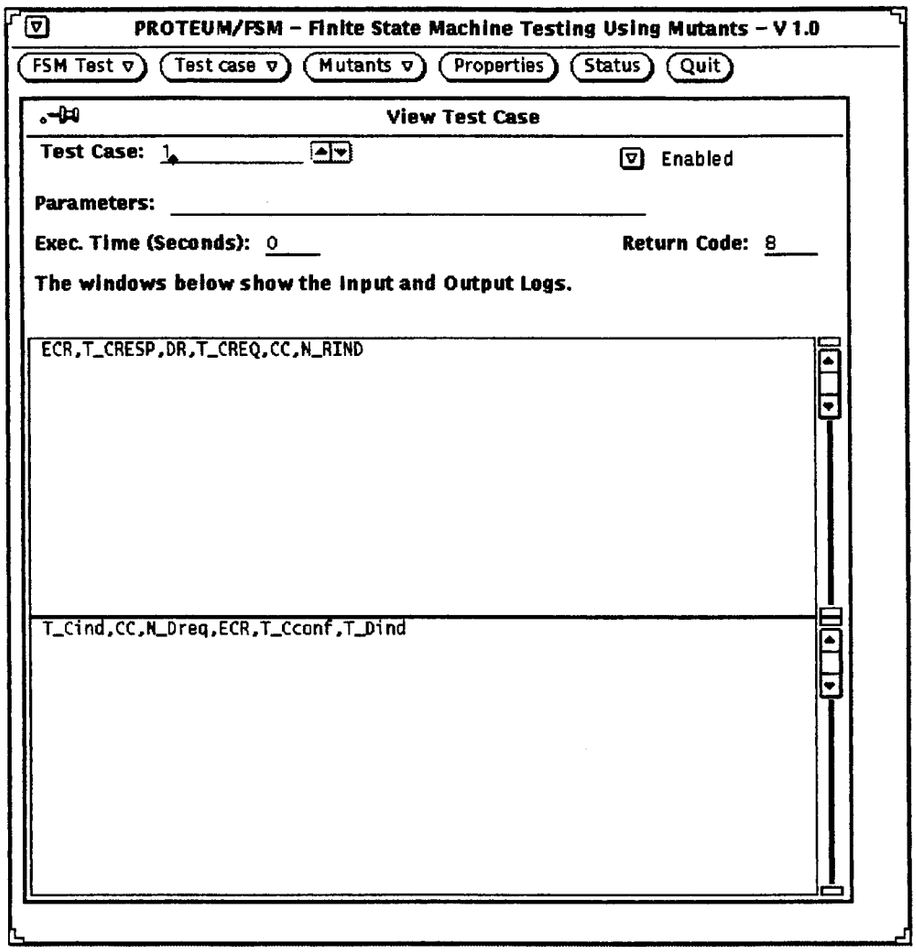


Figura 7.7 - Visualização de um Caso de Teste

Depois de inseridos um ou mais casos de teste, pode-se então executar os mutantes, ou seja, aplicar as seqüências de teste às MEFs mutantes. Para tanto, deve ser selecionada a opção *Execute*, que é uma das opções do botão *Mutants* do Menu Principal. A ferramenta inicia a execução dos mutantes com os casos de teste que estão habilitados no momento, não repetindo a execução para casos de teste que já tenham sido usados anteriormente. O resultado da execução dos mutantes é analisado através da comparação da saída gerada por eles com a saída gerada pela MEF original. À medida que os mutantes vão sendo executados, a ferramenta exibe o número do mutante e, como essa tarefa pode ser demorada, o usuário pode cancelar a execução através do botão *Cancel*, sendo mantidas as informações pertinentes aos mutantes já executados até então.

Depois de executados, os mutantes podem ser vistos através da opção *View*, Figura 7.8, que apresenta a LES da MEF original e do mutante, de forma a facilitar a tarefa do testador em analisar os mutantes vivos e marcá-los como equivalentes, quando for o caso. Essa janela fornece facilidades para que o testador selecione os tipos de mutantes que ele deseja visualizar. Por exemplo, se for selecionado apenas o botão *live*, ficam disponíveis para visualização somente os mutantes que permanecem vivos naquele momento. Além disso, com o botão *Equivalent* à direita do *status* do mutante, o testador pode marcar um mutante como equivalente após analisá-lo.

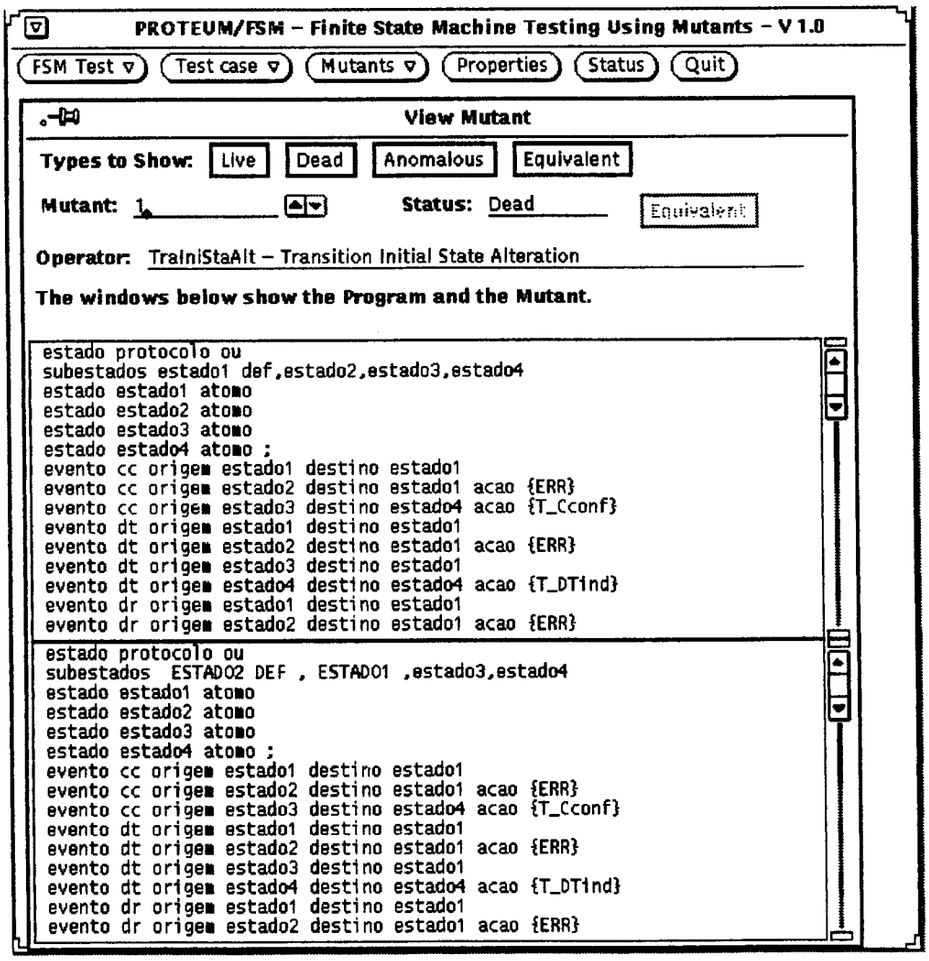


Figura 7.8 - Visualização de um Mutante

Quando desejado e, particularmente após a execução de mutantes, pode-se verificar o estado do teste através da opção *Status* do Menu Principal, Figura 7.9. A janela apresentada possui uma série de informações acerca desse teste, como por exemplo, o número de mutantes gerados até então, o número de mutantes equivalentes, o número de mutantes que permanecem vivos com base no conjunto de casos de teste utilizado na última execução, o número de mutantes que foram executados com aquele conjunto de casos de teste e o escore de mutação.

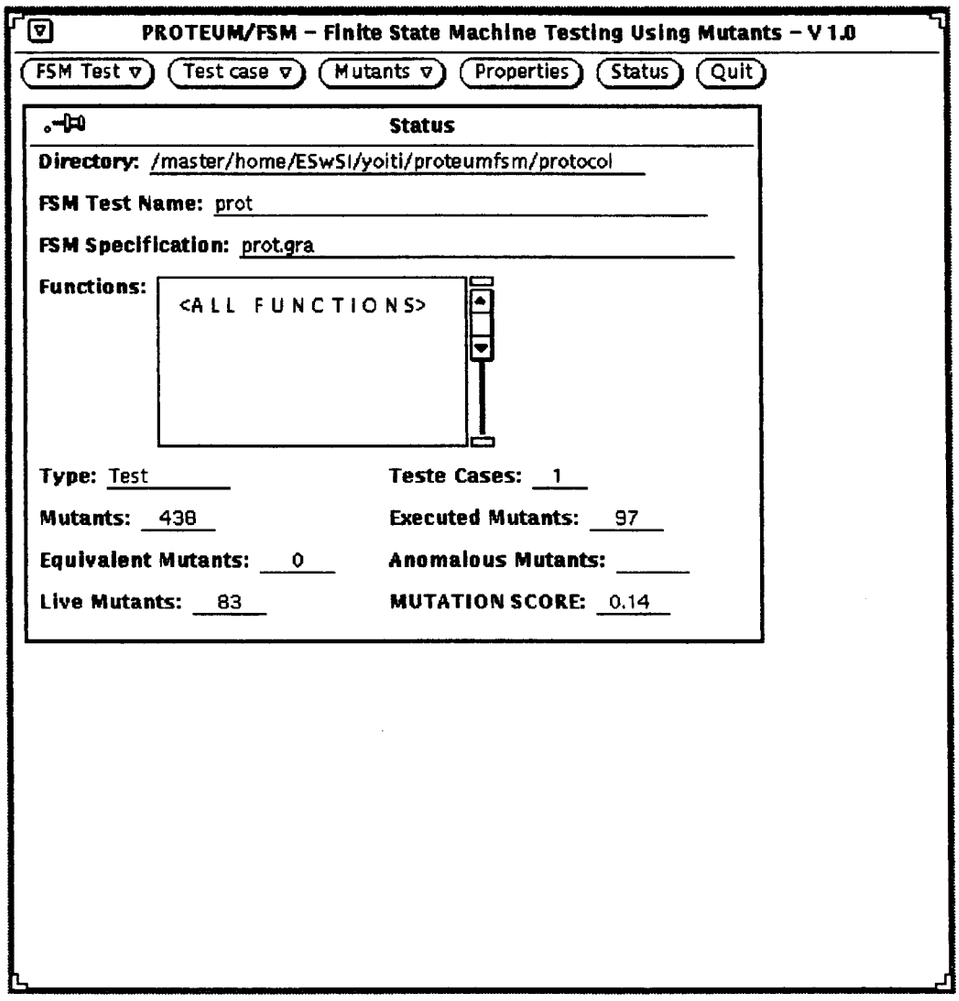


Figura 7.9 - Status da Sessão de Teste

O usuário pode encerrar a sessão de teste em andamento através da opção *Quit*. Então, ele pode escolher entre salvar as operações executadas na sessão de teste, através do botão *Save*, ou ignorá-las, não salvando nada da sessão realizada, através do botão *Abort*, ou então continuar a sessão de teste, através do botão *Cancel*.

7.5 Considerações Finais

Na implementação da ferramenta Proteum-RS/FSM foram reutilizados, na medida do possível e com as alterações necessárias, módulos de duas outras ferramentas: a Proteum/C (Delamaro, 1993), que apóia o teste de programas em C pelo critério Análise de Mutantes e o ambiente StatSim (Masiero et al, 1991), que apóia a edição e simulação de Statecharts.

Uma das características da ferramenta Proteum/C que facilitou o desenvolvimento da versão para Máquinas de Estados Finitos, foi o fato dela ser uma ferramenta multilinguagem. Essa característica permite que ela seja instanciada para outras linguagens de programação e, no caso, permitiu que ela fosse instanciada para um outro nível, o de especificação, uma vez que neste caso a linguagem que ela trata não corresponde a um programa, mas à especificação de uma MEF.

Os módulos do ambiente StatSim que foram reutilizados, correspondem àqueles que são responsáveis pela análise e simulação de Statecharts quando estes são fornecidos na forma textual, através da Linguagem de Especificação de Statecharts (LES). Salienta-se que a utilização do analisador e do simulador no contexto de Máquinas de Estados Finitos não exigiu nenhuma alteração na funcionalidade desses módulos. Eles apenas foram usados de forma restrita, pois para especificar uma MEF através da LES, é suficiente utilizar-se apenas um subconjunto dessa linguagem, uma vez que a técnica Statecharts é uma extensão das MEFs.

Portanto, a implementação da ferramenta Proteum-RS/FSM consistiu em construir um ambiente que integre facilidades de apoio ao teste de mutação e de análise de Máquinas de Estados Finitos, além de implementar o conjunto de operadores de mutação definidos neste trabalho.

Elaborada então a versão da ferramenta para Máquinas de Estados Finitos, instanciá-la para apoiar o teste de Statecharts corresponde, basicamente, à implementação dos operadores de mutação definidos para essa outra técnica. No caso de Redes de Petri, é preciso, além de implementar os operadores de mutação, dispor-se também de um simulador para Redes de Petri. Essas duas outras versões da ferramenta constituem objetivos de continuidade deste trabalho, sendo que a versão para Statecharts já se encontra em desenvolvimento.

CAPÍTULO 8

CONCLUSÕES

Este trabalho apresentou a extensão do critério Análise de Mutantes para o nível de especificações de sistemas. Essa extensão baseou-se fundamentalmente no estabelecimento de um paralelo da hipótese do programador competente e do efeito de acoplamento entre os níveis de programas e de especificação.

Um ponto chave para a aplicação do critério Análise de Mutantes é a definição de um conjunto de operadores de mutação. Assim, foram definidos os operadores de mutação para três técnicas de especificação do aspecto comportamental de Sistemas Reativos: Máquinas de Estados Finitos, Redes de Petri e Statecharts.

Para a definição desses operadores, inspirou-se em alguns trabalhos propostos na literatura. Um deles é o trabalho de Chow (1978) que estabelece três classes de erros para MEFs – erros de transferência, erros de operação e erros de estados extras/ausentes. O outro trabalho é de Weyuker et al (1994), onde são definidos alguns operadores de mutação utilizados para validar uma estratégia de teste definida pelos autores, para o teste de expressões booleanas. Considerou-se também o trabalho de Agrawal et al (1989), onde são definidos operadores de mutação para a linguagem C, extraindo-se desse trabalho alguns casos pertinentes ao contexto de especificações quanto ao uso de variáveis e condições.

Na definição dos operadores de mutação para Máquinas de Estados Finitos considerou-se como base o trabalho de Chow (1978) e as classes de erros definidas por ele. Na definição dos operadores para Redes de Petri, embora não

haja uma relação direta, inspirou-se também nessa classificação proposta por Chow, visto que para as Redes de Petri não existe uma classificação semelhante.

Para a técnica Statecharts os operadores de mutação foram separados em três classes distintas: 1) operadores que abordam aspectos relacionados às Máquinas de Estados Finitos, isto é, estados tipo OR e que são semelhantes àqueles definidos para as MEFs, mas com as adaptações necessárias; 2) operadores que tratam do uso de variáveis e condições, relacionados às Máquinas de Estados Finitos Estendidas; e 3) operadores que exploram características intrínsecas à técnica Statecharts, relativos a história, broadcastaing e paralelismo. Como Statecharts permite a especificação do sistema de forma hierárquica, definiram-se três estratégias de teste que abstraem os componentes de uma especificação Statecharts, permitindo que eles sejam testados isoladamente como MEFs. Para cada uma dessas estratégias foi definido um algoritmo que abstrai os componentes OR, no caso da Estratégia Básica, ou os componentes AND, no caso da Estratégia Baseada em Paralelismo, ou os componentes que possuem o símbolo de história associado, no caso da Estratégia Baseada em História.

Uma característica do critério Análise de Mutantes é o grande número de mutantes gerados, o que leva a altos custo e tempo de processamento, fatores que muitas vezes inviabilizam a sua utilização. Para minimizar esses problemas, assim como existe para o nível de programas, foram definidos dois critérios de mutação alternativa: a Mutação Aleatória, onde é gerada uma porcentagem dos mutantes para cada operador de mutação e a Mutação Restrita, onde são selecionados apenas alguns operadores para a geração dos mutantes.

A proposta de utilização da Análise de Mutantes no teste de especificações apresenta as seguintes vantagens:

- pode-se adaptar as atividades de teste às características do ambiente à medida que os operadores melhor cartacterizem os erros comumente cometidos pelos projetistas;
- pode-se conduzir as atividades de teste de forma incremental, selecionando primeiro um subconjunto de operadores e prosseguindo com as atividades de acordo com a criticalidade da aplicação e os recursos e tempo disponíveis;

- pode-se quantificar a atividade de teste, de maneira objetiva, através do escore de mutação.

Além dessas vantagens que são decorrentes das próprias características do critério Análise de Mutantes, pode-se citar alguns pontos mais específicos para cada uma das três técnicas utilizadas:

- no caso de Máquinas de Estados Finitos: atua como uma forma complementar de teste em relação aos métodos de geração de seqüências de teste uma vez que estes exigem que as MEFs satisfaçam alguns requisitos para sua aplicação e na prática, nem sempre esses requisitos são satisfeitos;
- no caso de Redes de Petri: atua como uma forma de auxílio ao testador na elaboração de seqüências de transições para o teste de especificações baseadas nessa técnica;
- no caso de Statecharts: além de atuar como uma forma de auxílio ao testador na elaboração de casos de teste, este pode ser abordado por partes, testando-se cada componente da especificação separadamente, com a aplicação das estratégias de teste propostas.

Para as Máquinas de Estados Finitos e Redes de Petri a aplicação da Análise de Mutantes foi validada através de experimentos realizados manualmente, onde se pode constatar as vantagens de utilização do critério e também das formas de mutação alternativa propostas. No caso do experimento realizado no contexto de MEFs observou-se que a seqüência de teste gerada pelo método W, por exemplo, não distinguiu os mutantes que exploraram o fato da MEF não ser completamente especificada, pois um dos requisitos exigidos para a aplicação desse método é que a MEF satisfaça a propriedade de especificação completa.

Dada a relevância da automatização da atividade de teste, em particular para a aplicação do critério Análise de Mutantes no contexto de Sistemas Reativos, foi especificada e implementada a ferramenta Proteum-RS/FSM, que dá suporte ao teste de Máquinas de Estados Finitos. Aspectos da instanciação dessa ferramenta para apoiar o teste de Redes de Petri e Statecharts foram também discutidos neste trabalho.

A proposta feita neste trabalho deve ainda ser explorada preferencialmente em problemas reais, com o objetivo de validar os mecanismos e ferramental aqui propostos, além de permitir o aprimoramento dos conjuntos de operadores de mutação e a definição de uma taxonomia de erros para as técnicas de especificação utilizadas.

O trabalho realizado está em sintonia com as pesquisas conduzidas atualmente pelo Grupo de Engenharia de Software do ICMSC, abrindo perspectivas para outros trabalhos e contribuindo para as atividades de pesquisa e de ensino do grupo. Certamente contribuirão para a consolidação das atividades de cooperação técnico-científica que já são desenvolvidas entre o ICMSC e o Departamento de Computação da UFSCar, local de trabalho da autora.

8.1 Contribuições deste Trabalho

Em seguida, relacionam-se algumas contribuições deste trabalho:

- Definição, aplicação e avaliação do uso do critério de teste Análise de Mutantes em nível de especificações;

- Definição formal dos operadores de mutação para as técnicas de especificação Máquinas de Estados Finitos, Redes de Petri e Statecharts;

- Definição de três estratégias de teste para Statecharts que, através de processos de abstração, permitem selecionar, por nível, componentes que possibilitam que o teste seja realizado por partes e de acordo com as características dos componentes abstraídos;

- Definição de critérios de mutação alternativa que visam à diminuição do número de mutantes gerados e, conseqüentemente, à diminuição do custo e tempo de aplicação do critério;

- Especificação e implementação da ferramenta Proteum-RS/FSM que apóia o teste de Máquinas de Estados Finitos pelo critério Análise de Mutantes e especificação das versões para Redes de Petri e Statecharts;

- Padronização de conceitos e critérios no que diz respeito à atividade de teste durante as fases de desenvolvimento de software;

- Em relação a alguns trabalhos encontrados na literatura, na área de especificação e teste de protocolos de comunicação, e que propõem o uso da Análise de Mutantes como uma forma de teste de especificações baseadas principalmente em Máquinas de Estados Finitos, este trabalho contribui na caracterização dos erros e definição formal dos operadores de mutação, além da aplicação efetiva do critério em nível de especificação. Dentre os trabalhos encontrados, comentados no Capítulo 3, nenhum deles aplica o critério efetivamente em nível de especificação. As mutações são identificadas na especificação, mas são aplicadas na implementação correspondente, possibilitando, em decorrência, que novos erros sejam introduzidos nesse processo.

8.2 Trabalhos Futuros

Em seguida, relacionam-se aspectos que abrem perspectivas de continuidade do trabalho proposto nesta tese:

- Implementar e acoplar ao conjunto de operadores de mutação da ferramenta Proteum-RS/FSM, a implementação dos operadores definidos para a técnica Statecharts que abordam aspectos de Máquinas de Estados Finitos Estendidas, produzindo com isso uma versão que apóie tais aspectos e que pode ser denominada Proteum-RS/EFSM.

- Implementar a versão Proteum-RS/PN, para o teste de especificações baseadas em Redes de Petri.

- Implementar a versão Proteum-RS/ST, para o teste de especificações baseadas em Statecharts.

- Acoplar o editor gráfico do ambiente StatSim na versão Proteum-RS/FSM de modo a facilitar a edição de Máquinas de Estados Finitos.

- Explorar a questão de equivalência no contexto das três técnicas de especificação, com o objetivo de definirem-se heurísticas que auxiliem o testador na decisão da equivalência dos mutantes em relação à especificação original.

- Definir e conduzir estudos empíricos para cada uma das três técnicas de especificação, utilizando as ferramentas nas três versões propostas, com o objetivo de avaliar a eficácia dos operadores de mutação, viabilizando uma melhor estratégia de aplicação dos mesmos no teste incremental.

REFERÊNCIAS BIBLIOGRÁFICAS

Acree, A.; Budd, R.; DeMillo, R.A.; Lipton, R.; Sayward, F.G. "Mutation Analysis", *Technical Report GIT-ICS-79/08*, Georgia Institute of Technology, Setembro, 1979.

Acree, A. "On Mutation", *Ph.D. thesis*, School of Information and Computer Science, Georgia Institute of Technology, Agosto, 1980.

Agrawal, H.; DeMillo, R.A.; Hathaway, R.; Hsu, Wm.; Hsu, W.; Krauser, E.; Martin, R.J.; Mathur, A.P.; Spafford, E. "Design of Mutant Operators for the C Programming Language", *Technical Report SERC-TR-41-P*, Software Eng. Research Center, Purdue University, Março, 1989.

Batista, J.E.S. "Um Editor Gráfico para Statecharts", *Dissertação de Mestrado*, ICMSC-USP, 1991.

Boaventura, I.A.G. "Propriedades Dinâmicas de Statecharts", *Dissertação de Mestrado*, ICMSC-USP, 1992.

Bochmann, G.v.; Das, A.; Dssouli, R.; Dubuc, M.; Ghedamsi, A.; Luo, G. "Fault Models in Testing", *IV Protocol Test Systems*, Elsevier Science Publishers B. V., North-Holland, 1992.

Bochmann, G.v.; Petrenko, A. "Protocol Testing: Review of Methods and Relevance for Software Testing", in *Proceedings of the ISSTA'1994 - International Symposium on Software Testing and Analysis*, ACM - Software Engineering Notes, pp. 109-124, 1994.

Bolognesi, T.; Brinksma, E. "Introduction to the ISO Specification Language LOTOS", *Computer Networks and ISDN Systems*, Vol.14, pp.25-59, 1987.

Boussinot, F.; Simone, R. "The ESTEREL Language", in Proceedings of the IEEE, Vol.79, N.9, Setembro, 1991.

Budd, T.A.; DeMillo, R.A.; Lipton, R.J.; Sayward, F.G. "Theoretical and Empirical Studies on Using Prog Mutation to Test the Functional Correctiness of Prog.", in *7th ACM Symposium on Principles of Programming Languages*, Janeiro, 1980.

Budd, T.A. "Mutation Analysis: Ideas, Examples, Problems and Prospects", *Computer Program Testing*, North-Holand Publishing Company, pp.129-148, 1981.

Budkowski, S.; Dembinski, P. "An Introduction to Estelle: A Specification Language for Distributed Systems", *Computer Networks and ISDN Systems*, Vol.14, N.1, pp.3-23, 1987.

Cangussu, J.W.L.; Penteado, R.D.; Masiero, P.C.; Maldonado, J.C. "Validation of Statecharts Based on Programmed Execution", in *Proc. 7th International Conference on Computer and Information*, Peterborough, Ontario, Canada, 1995.

Cangussu, J.W.L. "Execução Programada de Statecharts", *Dissertação de Mestrado*, ICMSC-USP, 1993.

Chaim, M.L. "POKE-TOOL - UMa Ferramenta para Suporte ao Teste Estrutural de Programas Baseados em Análise de Fluxo de Dados", *Dissertação de Mestrado*, DCA/FEE/UNICAMP, Campinas, SP, Abril, 1991.

Choi, B.J.; Mathur, A.P.; Pattison, B. "pMothra: Scheduling Mutants for Execution on a Hipercube", *Third Symposium on Software Testing, Analysis and Verification*, Key West, Dezembro, 1989a.

Choi, B.J.; DeMillo, R.A.; Krawser, E.W.; Martin, R.J.; Mathur, A.P.; Offut, A.J.; Pan, H.; Spafford, E.H. "The Mothra Tool Set", in *Proceedings of the 22nd Hawaii International Conference on Systems and Software*, Kona, Hawaii, Janeiro, 1989b.

Choi, B.; Mathur, A.P. "High-Performance Mutation Testing", *Journal of Systems and Software*, Vol.20, pp. 135-152, 1993.

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

Chung, A.; Sidhu, D. "Fault Coverage of Probabilistic Test Sequences", in *Proceedings of the IFIP TC6 Third International Workshop on Protocol Test Systems*, North-Holland, pp. 305-316, 1991.

Coleman, D.; Hayes, F.; Bear, S, "Introducing Objectcharts or How to Use Statecharts in Object-Oriented Design", *IEEE Transactions on Software Engineering*, Vol.18, N.1, pp.9-18, Janeiro, 1992.

Davis, A.M. "A Comparison of Techniques for the Specification of External System Behavior", *Communications of the ACM*, Vol.31, N. 9, Setembro, 1988.

Delamaro,M.E.; Maldonado, J.C. "Uma Visão sobre a Aplicação da Análise de Mutantes", *Notas do ICMSC-USP*, N.133, Março, 1993.

Delamaro, M.E. "Proteum - Um Ambiente de Teste Baseado na Análise de Mutantes", *Dissertação de Mestrado*, ICMSC-USP, Outubro, 1993.

Delamaro,M.E.; Maldonado, J.C. "Proteum: A Tool for the Assessment of Test Adequacy for C Programs", in *Proceedings of PCS'96 - Conference of Performability in Computing Systems*, Brunswick, N.J., Julho, 1996a.

Delamaro,M.E.; Maldonado, J.C. "Proteum Version 1.2C - Running Program Tests Through Shell Scripts", (a ser publicado), 1996b.

Delamaro,M.E.; Maldonado, J.C.; Mathur, A.P. "Integration Testing Using Interface Mutation", *ISSRE'96 - 7th International Symposium on Software Reliability Engineering*, N.Y., aceito para publicação, 1996.

DeMillo, R.A.; Lipton, R.J.; Sayward, F.G. "Hints on Test Data Selection: Help for the Practicing Programmer", *Computer*, Vol.11(4), pp.34-41, 1978.

DeMillo, R.A. "Mutation Analisys as a Tool for Software Quality Assurance", in *Proc. of COMPSAC 80*, Chicago-IL, Outubro, 1980.

DeMillo, R.A. *Software Testing and Evaluation*, The Benjamin/Cummings Publishing Company, Inc, 1987.

DeMillo, R.A.; Guindi, D.S.; King, K.N.; McKrake, W.N.; Offutt, A.J. "An Extended Overview of the Mothra Testing Environment", in *Proceedings of the Second Workshop on Software Testing, Verification and Analysis*, Banff, Canada, 1988.

DeMillo, R.A.; Offutt, A.J. "Constrained-Based Automatic Teste Data Generation", *IEEE Transactions on Software Engineering*, Vol.17, N.9, pp.900-910, Setembro, 1991.

Duncan, I.M.M.; Robson, D.J. "Ordered Mutation Testing", *ACM SIGSOFT Software Engineering Notes*, Vol.15, N.2, pp.29-30, Abril, 1990.

Fabbri, S.C.P.F.; Maldonado, J.C.; Masiero, P.C.; Delamaro, M.E. "Análise de Mutantes Baseada em Máquinas de Estado Finito", in *Anais do 11º Simpósio Brasileiro de Redes de Computadores*, Campinas, Maio, 1993.

Fabbri, S.C.P.F.; Delamaro, M.E.; Maldonado, J.C.; Masiero,P.C. "Proteum/FSM: Especificação de uma Ferramenta para Apoiar a Validação de Máquinas de Estado Finito pelo Critério Análise de Mutantes", in *Anais do 12º Simpósio Brasileiro de Redes de Computadores*, pp.284-302, Curitiba, Maio, 1994a.

Fabbri, S.C.P.F.; Maldonado, J.C.; Delamaro, M.E.; Masiero, P.C. "Mutation Analysis Testing for Finite State Machines", in *Proc. ISSRE'94 - Fifth International Symposium on Software Reliability Engineering*, pp.220-229, California, Novembro, 1994b.

Fabbri, S.C.P.F.; Maldonado, J.C.; Masiero, P.C.; Delamaro, M.E. "Aplicação do Critério Análise de Mutantes na Validação de Especificações Baseadas em Redes de Petri", in *Anais do VIII Simpósio Brasileiro de Engenharia de Software*, Curitiba, Outubro, 1994c.

Fabbri, S.C.P.F.; Maldonado, J.C.; Masiero, P.C.; Delamaro, M.E.; Wong, E. "Mutation Testing Applied to Validate Specifications Based on Petri Nets", in *Proc. FORTE'95 - 8th International IFIP Conference on Formal Description Techniques for Distributed Systems and Communications Protocols*, Montreal, Canada, Outubro, 1995a.

Fabbri, S.C.P.F.; Maldonado, J.C.; Delamaro, M.E.; Masiero, P.C. "Proteum/FSM - Uma Ferramenta para Apoiar a Validação de Máquinas de Estados Finitos pelo Critério Análise de Mutantes", in *Anais do IX Simpósio Brasileiro de Engenharia de Software*, pp.475-478, Recife, Pernambuco, Outubro, 1995b.

Fleyshgakker, V.N.; Weiss, S.N. "Efficient Mutation Analysis: A New Approach", in *Proceedings of the ISSTA'94 - International Symposium on Software Testing and Analysis, ACM - Software Engineering Notes*, pp.185-195, 1994.

Fortes, R.P.M. "Uma Ferramenta de Apoio à Utilização de Statecharts para Especificação do Comportamento de Sistemas de Tempo Real Complexos", *Dissertação de Mestrado*, ICMSC - USP, 1991.

Friedman, A.D. *Logical Design of Digital Systems*, Computer Science Press, 1975.

Fujiwara, S.; Bochmann, G.V.; Khendek,F.; Amalou, M.; Ghedamsi, A. "Test Selection Based on Finite State Models", *IEEE Transaction on Software Engineering*, Vol. 17, N. 6, Junho, 1991.

Furbach, U. "Formal Specification Methods for Reactive Systems", *Journal of Systems and Software*, N.21, pp.129-139, 1993.

Gabos, D.; Stiubiener, S. "Aspectos de Metodologia de Geração de Seqüências de Teste para Protocolos de Comunicação de Dados", in *Anais 8º Simpósio de Redes de Computadores*, 1990.

Gane, C.; Sarson, T. *Structured Systems Analysis: Tools and Techniques*, New Jersey, Prentice-Hall, 1979.

Gill, A. *Introduction to the Theory of Finite-State Machines*, New York, McGraw-Hill, 1962.

Gonenc, G. "A Method for the Design of Fault-Detection Experiments", *IEEE Transaction on Computer*, Vol C-19, pp 551-558, Junho, 1970.

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

Harel, D. "Statecharts: On the Formal Semantics of Statecharts", in *Proc. 2nd IEEE Symposium on Logic in Computer Science*, Ithaca, New York, 1987b.

Harel, D.; Lachover, H.; Naamad, A.; Pnueli, A.; Politi, M.; Sherman, R.; Shtull-Trauring, A.; Trakhtenbrot, M. "STATEMATE: A Working Environment for the Development of Complex Reactive Sistems", *IEEE Transactions on Software Engineering*, V.16, N.4, pp.403-414, 1990.

Harel, D. "Biting the Silver Bullet - Toward a Brighter Future for Systems Development", *Computer IEEE*, pp.8-20, Janeiro, 1992.

Hoare, C.A.R. *Communicating Sequential Processes*, Prentice-Hall, Englewood-Cliffs, New Jersey, 1984.

Horgan, J.R. "ATAC - Automatic Test Coverage Analysis for C Programas", *MRE 2E-362/x4338*, Bellcore, Junho, 1990.

Horgan, J.R.; Mathur, A.P. "Assessing Testing Tools in Research and Education", *IEEE Software*, Vol. 9, N. 3, Maio, 1992.

Howden, W.E. "Weak Mutation Testing and Completeness of Test Sets", *IEEE Transactions on Software Engineering*, Vol. SE-8, N. 4, Julho, 1982.

IBM Corporation, Data Processing Division *HIPO: A Design Aid and Documentation Technique*, White Plains, New York, 1974.

Jackson, M. *System Development*, Prentice-Hall International, Englewood Cliffs, 1983.

Jones, C.B. *Systematic Software Development Using VDM*, Prentice-Hall, 1986.

Kohavi, Z. *Switching and Finite Automata Theory*, New York, McGraw-Hill, 1978.

Krauser, E.W. "Compiler Integrated Testing", *Ph.D. thesis*, Purdue University, Dezembro, 1991.

Krauser, E.W.; Mathur, A.P.; Rego, V. "High Performance SoftwareTesting on SIMD Machines", *IEEE Transactions on Software Engineering*, Vol. 17, N. 5, pp.403-423, Maio 1991.

Leveson, N.G.; Heimdahl, M.; Hildreth, H.; Reese, J.; Ortega, R. "Experiences Using Statecharts for a System Requirement Specification", in *Proceegings of International Workshop on Software Specification and Design*, Lake Como, Italy, Outubro, 1991.

Leveson, N.G.; Heimdahl, M.P.E.; Hildreth, H.; Reese, J.D. "Requirements Specification for Process-Control Systems", *IEEE Transactions on Software Engineering*, Vol.20, N.9, pp.684-707, Setembro, 1994.

LNCS - Lecture Notes in Computer Science - Application and Theory of Petri Nets, N.815, 1994.

LNCS - Lecture Notes in Computer Science - Application and Theory of Petri Nets, N.935, 1995.

Luo, G.; Petrenko, A.; Bochmann, G.v. "Test Selection Based on Communicating Nondeterministic Finite State Machines using a Generalized Wp-Method", *IEEE Transactions on Software Engineering*, Vol.SE-20, N.2, 1994.

Maldonado, J.C. "Critérios Potenciais Usos: Uma Contribuição ao Teste Estrutural de Software", *Tese de Doutorado*, FEE/Unicamp, Campinas - SP, 1991.

Maldonado, J.C.; Delamaro, M.E.; Souza, S.R.S.; Wong, W.E. "A Comparison of Constrained Mutation Testing in C and Fortran", submetido, 1996.

Marshall, A.C.; Hedley, D.; Riddell, I.J.; Hennell, M.A. "Static Dataflow-Aided Weak Mutation Analysis (SDAWM)", *Information and Software Technology*, Vol.32, N.1, Janeiro/Fevereiro, 1990.

Masiero,P.C.; Fortes,R.P.M.; Batista Neto,J.E.S. "Edição e Simulação do Aspecto Comportamental de Sistemas de Tempo Real", in *Anais do XI Congresso Nacional da SBC, XVIII SEMISH*, Santos, pp. 45-61, Agosto, 1991.

Masiero, P.C.; Maldonado, J.C.; Boaventura, I.G. "A Reachability Tree for Statecharts and Analysis of Some Properties", *Information and Software Technology*, Vol.36 (10), pp.615-624, 1994a.

Masiero, P.C.; Oliveira, M.C.; Germano, F.S.; Pierri, G. "Authoring and Searching in Dynamically Growing Hypertext Databases", *Hypermedia*, London, V.6, N.2, pp.124-148, 1994b.

Mathur, A.P.; Krauser, E.W. "Modeling Mutation on Vector Processor", in *Proceedings of the 2nd Workshop on Software Testing, Verification and Analysis*, Banff, Canada, 1988.

Mathur, A.P.; Wong, W.E. "Evaluation of the Cost of Alternate Mutation Strategies", in *Anais VII SBES - Simpósio Brasileiro de Engenharia de Software*, Rio de Janeiro, RJ, pp.320-335, Outubro, 1993.

Mathur, A.P.; Wong, W.E. "An Empirical Comparison of Data Flow and Mutation-Based Test Adequacy Criteria", *Software Testing, Verification and Reliability*, V.4, pp.9-31, 1994.

Milner, A.J.R.G. "A Calculus of Communicating Systems", *Lecture Notes in Computer Science 92*, Springer-Verlag, 1980.

Morell, L.J. "A Theory of Fault-Based Testing", *IEEE Transactions on Software Engineering*, Vol.18, N.8, Agosto, 1990.

Murata, T. "Petri Nets: Properties, Analysis and Applications", in *Proceedings of the IEEE*, Vol.77, N.4, pp.541-580, Abril, 1989.

Myers, G. *The Art of Software Testing*, Wiley, New York, 1979.

Naito, S.; Tsunoyama, M. "Fault Detection for Sequential Machines by Transition-Tours", in *Proceedings FTCS - Fault Tolerant Comput. Systems*, pp. 238-243, 1981.

Nakazato, K.K.; Maldonado, J.C.; Fabbri, S.C.P.F.; Masiero, P.C. "Propriedades de Máquinas de Estados Finitos Relevantes para Critérios de Geração de Seqüências de Teste", *Relatório Técnico do ICMSC-USP, N.27*, Outubro, 1994a.

Nakazato, K.K.; Maldonado, J.C.; Fabbri, S.C.P.F.; Masiero, P.C. "Seqüências Básicas de Máquinas de Estados Finitos: Aspectos Teóricos e de Implementação", *Relatório Técnico do ICMSC-USP*, N.28, Outubro, 1994b.

Nakazato, K.K.; Maldonado, J.C.; Fabbri, S.C.P.F.; Masiero, P.C. "Aspectos Teóricos e de Implementação de Critérios de Geração de Seqüências de Teste Baseados em Máquinas de Estados Finitos", *Relatório Técnico do ICMSC-USP, N.25*, Outubro, 1994c.

Nakazato, K.K.; Maldonado, J.C.; Fabbri, S.C.P.F.; Masiero, P.C. "Aspectos sobre a Equivalência de Máquinas de Estados Finitos", *Relatório Técnico do ICMSC-USP*, (a ser publicado), 1996.

Nakazato, K.K. "MGASET: Módulo de Geração de Seqüências de Teste" *Dissertação de Mestrado*, ICMSC-USP, Janeiro, 1995.

Notomi, M.; Murata, T. "Hierarchical Reachability Grapf of Bounded Petri Nets for Concurrent-Software Analysis", *IEEE Transactions on Software Engineering*, Vol.20, N.5, pp.325-336, Maio, 1994.

Offutt, A.J.; King, K.N. "A Fortran 77 Interpreter for Mutation Analysis", in *Proceedings of the SIGPLAN 87 Symposium on Interpreters and Interpretive Techniques*, St. Paul, Minnesota, Julho, 1987.

Offutt, A.J. "Investigations of the Software Testing Coupling Effect", *ACM Transactions on Software Engineering Methodology*, 1(1), pp.3-18, Janeiro, 1992.

Offutt, A.J.; Rothermel, A.J. "An Experimental Evaluation of Selective Mutation", *15th International Conference on Software Engineering*, Baltimore, Maryland, Maio, 1993.

Offutt, A.J.; Lee, S. "An Empirical Evaluation of Weak Mutation", *IEEE Transactions on Software Engineering*, Vol. 20, N. 5, pp. 337-344, Maio, 1994.

Offutt, A.J.; Rothermel, G.; Untch, R.H.; Zapf, C. "An Experimental Determination of Sufficient Mutant Operators", *ACM Transactions on Software Engineering Methodology*, accepted for publication, 1996.

Peterson, J.L. *Petri Nets*, Computing Surveys, Vol.9, N. 3, Setembro, 1977.

Peterson, J.L. *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Englewood Cliffs, N.J, 1981.

Petrenko, A. "Checking Experiments with Protocol Machines", in *Proceedings IWPTS'91 - International Workshop on Protocol Test Systems*, 1991.

Petrenko, A.; Yevtushenko, N.; Lebedev, A.; Das, A. "Nondeterministic State Machines in Protocol Conformance Testing", in *Proceedings IWPTS'93 - International Workshop on Protocol Test Systems*, 1993.

Petrenko, A.; Bochmann, G.v. "On Fault Coverage of Tests for Finite State Specifications", http://www.iro.umontreal.ca/pub/teleinfo/TRs/Petr96b.ps.gz, 1996.

Pressman, R.S. *Software Engineering - A Practitioner's Approach*, (3rd edition), McGraw-Hill, 1992.

Price, A.M.; Zorzo, A. "Visualizando o Fluxo de Controle de Programas", in *Anais IV SBES - Simpósio Brasileiro de Engenharia de Software*, Águas de São Pedro, SP, Outubro, 1990.

Probert, R.L.; Guo, F. "Mutation Testing of Protocols: Principles and Preliminary Experimental Results", in *Proceedings of the IFIP TC6 Third International Workshop on Protocol Test Systems*, North-Holland, pp. 57-76, 1991.

Rapps, S.; Weyuker, E.J. "Data Flow Analysis Techniques for Test Data Selection", in *Proc. International Conference on Software Engineering*, Tokio, pp.272-278, Setembro, 1982.

Rapps, S.; Weyuker, E.J. "Selecting Software Testing Data Using Data Flow Information", *IEEE Transactions on Software Engineering*, Vol.SE-11, pp.367-375, Abril, 1985.

Sabnani, K.K.; Dahbura, A.T. "A Protocol Testing Procedure", *Computer Networks and ISDN Syst.*, Vol. 15, N. 4, pp. 285-297, 1988.

Semmens, L.T.; France, R.B.; Docker, T.W.G. "Integrated Structured Analysis and Formal Specification Techniques", *The Computer Journal*, V.35, N.6, 1992.

Sidhu, D.P.; Leung, T.K. "Formal Methods for Protocol Testing: A Detailed Study", *IEEE Transactions on Software Engineering*, Vol.SE-15, N.4, pp.413-425, Abril, 1989.

Sowmya, A. "A Statecharts-Based Specification and Verification of Real-Time Scheduling Systems", in *Proceedings of International Workshop on Real-Time Programming Bruges*, Belgium, 1992.

Souza, S.R.S. "Avaliação do Custo e Eficácia do Critério Análise de Mutantes na Atividade de Teste de Programas", *Dissertação de Mestrado*, ICMSC-USP, Junho, 1996.

Spivey, J.M. *Introducing Z: A Specification Language and its Formal Semantics*, Cambridge University Press, 1988.

Stotts, P.D.; Furuta,R. "Petri-Net-Based Hypertext: Document Structure with Browsing Semantics", *ACM Transactions on Information Systems*, Vol.7, N.1, pp.3-29, Janeiro, 1989.

Tan, Q.M.; Petrenko, A.; Bochmann, G.v. "A Test Generation Tool for Specifications in the Form of State Machines", http://www.iro.umontreal.ca/pub/teleinfo/TRs/P1016.ps.gz, 1996.

Tanenbaum, A.S. *Computer Networks*, (2nd. edition), Prentice Hall, 1989.

Wang, C.J.; Liu, M.T. "Generating Test Cases for EFSM with Given Fault Model", *IEEE INFOCOM'93 - 12th Annual Joint Conference of the IEEE Computer and Communications Societies*, Vol.2, pp. 774-781, 1993.

Weiss, S.N.; Fleyshgakker, V.N. "Improved Serial Algorithms for Mutation Analysis", in *Proceedings of the ISSTA'93 - International Symposium on Software Testing and Analysis*, ACM - Software Engineering Notes, pp.149-158, 1993.

Weyuker, E.; Goradia, T.; Singh, A. "Automatically Generating Test Data from a Boolean Specification", *IEEE Trans. on Software Engineering*, Vol. 20, N. 5, pp.353-363, Maio, 1994.

Wing, J.M. "A Specifier's Introduction to Formal Methods", *IEEE Computer*, pp.8-22, Setembro, 1990.

Wong, W.E. "On Mutatioin and Data Flow", *Thesis*, SERC-TR-149-P, Software Engineering Research Center, Purdue University, Dezembro, 1993.

Wong, W.E.; Maldonado, J.C.; Delamaro, M.E.; Mathur, A.P. "Constrained Mutation in C Programs", in *Anais do VIII SBES - Simpósio Brasileiro de Engenharia de Software*, Curitiba, PR, Outubro, pp. 439-452, 1994a.

Wong, W.E.; Maldonado, J.C.; Mathur, A.P. "Mutation versus All-uses: An Empirical Evaluation of Cost, Strength and Effectiveness", in *Proceedings of the First IFIP/SQI International Conference on Software Quality and Productivity*, Hong Kong, Novembro, 1994b.

Woodward, M.R. "Mutation Testing - its Origin and Evolution", *Information and Software Technology*, Vol. 35, N. 3, pp. 163-169, Março, 1993.

Yao, M.; Petrenko, A.; Bochmann, G.v. "A Structural Analysis Approach to the Evaluation of Fault Coverage for Protocol Conformance Testing", in *Proceedings of the 7th IFIP WG 6.1 International Conference on Formal Description Techniques*, FORTE'94, pp.399-414, 1994a.

Yao, M.; Petrenko, A.; Bochmann, G.v. "Fault Coverage Analysis in Respect to a FSM Specification", *IEEE INFOCOM'94*, Toronto, Canada, pp.768-775, 1994b.

Yevtushenko, N.; Lebedev, A.; Petrenko, A. "On the Checking Experiments with Nondeterministic Automata", *Automatic Control and Computer Sciences*, Allerton-Press, Inc., N.Y., Vol.25, N.6, 1991.

Zheng, Y.; Pong, M.C. "Using Statecharts to Model Hipertext", in *Proceedings of ACM ECHT Conference*, Milano, pp.242-250, Novembro/Dezembro, 1992.

APÊNDICE A

GRAMÁTICA DA LINGUAGEM DE ESPECIFICAÇÃO DE STATECHARTS - LES

Neste apêndice apresenta-se a gramática da Linguagem de Especificação de Statecharts - LES (Fortes, 1991). A Seção A.1, contém a gramática completa, com as expressões necessárias para descrição de todos os aspectos da técnica Statecharts, como paralelismo, história, broadcasting, etc. Nas Seções A.2 e A.3, apresentam-se, respectivamente, os subconjuntos da LES com as expressões necessárias para a descrição das Máquinas de Estados Finitos e das Máquinas de Estados Finitos Estendidas.

A.1 LES para Descrição de Statecharts

< estadograma > ::=
    [< declarações > ';']
    < estados > ';'
    < transições > ';'

< declarações > ::=
    < declaração > { < declaração > }

< declaração > ::=
    < tipo-id > < lista-identificadores >

< tipo-id > ::= (**INT** | **BOOL** | **REAL** | **STRING** | **COND**)

< lista-identificadores > ::=
< identificador > { ',' < identificador > }

< estados > ::=
< descrição estado > { < descrição estado > }

< descrição estado > ::=
ESTADO < nome-estado > < tipo >

< tipo > ::=
OU < subestados ou > | **E** < subestados e > | **ATOMO**

< subestados ou > ::=
SUBESTADOS < nome-estado > [(**DEF** | **DEF/H** | **DEF/H***)]
{ ',' < nome-estado > }

< subestados e > ::=
SUBESTADOS < nome-estado > { ',' < nome-estado > }

< transições > ::=
< transição > { < transição > }

< transição > ::=
EVENTO < evento >
ORIGEM < estado-origem > { ',' < estado-origem > }
DESTINO < estado-destino > [(**/ H** | **/ H***)]
{ ',' < estado-destino > [(**/ H** | **/ H***)] }
[**AÇÃO** '{' < ações > '}']

< evento > ::=
< ev-ou > { or < ev-ou > }

```
< ev-ou > ::=
        < ev-e >  { and  < ev-e > }

< ev-e > ::=
        < ev-f >  [  '[' < exp > ']'  ]

< ev-f > ::=
          < evento-primitivo >
        | < true >  '(' < exp > ')'
        | < false >  '(' < exp > ')'
        | < changed >  '(' < exp > ')'
        | < exit >  '(' < nome-estado >  ')'
        | < entered >  '(' < nome-estado >  ')'
        | < t-out >  '(' < número >  < unid-tempo >  ')'

< exp > ::=
        < expS > [ ( = | > | < | <> | <= | >= ) < expS > ]

< expS > ::=
        [ + | - ] < termo > { ( + | - | or ) < termo > }

< termo > ::=
        < fator > { ( * | / | and ) < fator > }

< fator > ::=
        | < true >
        | < false >
        | < condição-primitiva >
        | < número >
        | < variável >
        | in  '(' < nome-estado > ')'
        | < not-yet >  '(' < evento > ')'
        | < current >  '(' < exp > ')'
        | not < exp >
        | '(' < exp > ')'
```

< ações > ::=
    < ação > { ';' < ação > }

< ação > ::=
      < identificador >
    | < variável > ':=' < exp >
    | clear '(' < nome-estado > [*] ')'

< estado-origem > ::= < nome-estado >

< estado-destino > ::= < nome-estado >

< nome-estado > ::=
    < identificador > ['.' < identificador >]

< evento-primitivo > ::= < identificador >

< condição-primitiva > ::= < identificador >

< variável > ::= < identificador >

Símbolos Terminais:

< true > ::= ([t T] [r R] [u U] [e E] | tr)

< false > ::= ([f F] [a A] [l L] [s S] [e E] | fs)

< changed > ::= (changed | ch)

< exit > ::= (exit | ex)

< entered > ::= (entered | en)

< not-yet > ::= (not-yet | ny)

< current > ::= (current | cr)

< identificador > ::= [a - z A - Z] [a - z A - Z 0 - 9] *

< número > ::= [0 - 9] +

< unid-tempo > ::= (seg | min | hr)

Observações:

[α] - a cadeia α é opcional

{ α } - repetição da cadeia α zero ou mais vezes

α | β - α ou β deve ser escolhida

A.2 LES para Descrição de Máquinas de Estados Finitos

Para a descrição de Máquinas de Estados Finitos utiliza-se um subconjunto restrito da LES original que contém, basicamente, as expressões necessárias para descrever um estado do tipo OR, os subestados que o compõem e as transições definidas na MEF.

< estadograma > ::=

< estados > ';'

< transições > ';'

< estados > ::=

< descrição estado > { < descrição estado > }

< descrição estado > ::=

ESTADO < nome-estado > < tipo >

< tipo > ::=

OU < subestados ou > | **ATOMO**

< subestados ou > ::=

SUBESTADOS < nome-estado > [**DEF**] { ',' < nome-estado > }

< transições > ::=

< transição > { < transição > }

< transição > ::=

EVENTO < evento >

ORIGEM < estado-origem >

DESTINO < estado-destino >

[**AÇÃO** '{' < ações > '}']

< evento > ::= < evento-primitivo >

< ações > ::=

< ação > { ';' < ação > }

< ação > ::= < identificador >

< estado-origem > ::= < nome-estado >

< estado-destino > ::= < nome-estado >

< nome-estado > ::= < identificador >

< evento-primitivo > ::= < identificador >

Símbolos Terminais:

< identificador > ::= [a - z A - Z] [a - z A - Z 0 - 9] *

A.3 LES para Descrição de Máquinas de Estados Finitos Estendidas

Para descrição de Máquinas de Estados Finitos Estendidas utiliza-se um subconjunto restrito da LES original que contém, além das expressões necessárias para a descrição de Máquinas de Estados Finitos, aquelas utilizadas para declarações e uso de variáveis. Ou seja, é um subconjunto da LES que não possui as expressões relativas ao paralelismo de estados, decomposição e aspectos de história.

< estadograma > ::=
    [< declarações > ';']
    < estados > ';'
    < transições > ';'

< declarações > ::=
    < declaração > { < declaração > }

< declaração > ::=
    < tipo-id > < lista-identificadores >

< tipo-id > ::= (**INT** | **BOOL** | **REAL** | **COND**)

< lista-identificadores > ::=
    < identificador > { ',' < identificador > }

< estados > ::=
    < descrição estado > { < descrição estado > }

< descrição estado > ::=
    ESTADO < nome-estado > < tipo >

< tipo > ::=
    OU < subestados ou > | **ATOMO**

< subestados ou > ::=

SUBESTADOS < nome-estado > [**DEF**] { ',' < nome-estado > }

< transições > ::=

< transição > { < transição > }

< transição > ::=

EVENTO < evento >

ORIGEM < estado-origem >

DESTINO < estado-destino >

[**AÇÃO** '{' < ações > '}']

< evento > ::=

< ev-f > ['[' < exp > ']']

< ev-f > ::=

< evento-primitivo >

| < true > '(' < exp > ')'

| < false > '(' < exp > ')'

| < changed > '(' < exp > ')'

< exp > ::=

< expS > [(= | > | < | <> | <= | >=) < expS >]

< expS > ::=

[+ | -] < termo > { (+ | - | or) < termo > }

< termo > ::=

< fator > { (* | / | and) < fator > }

< fator > ::=

| < true >

| < false >

| < condição-primitiva >

| < número >
| < variável >
| not < exp >
| '(' < exp > ')'

< ações > ::=
< ação > { ';' < ação > }

< ação > ::=
< identificador >
| < variável > ':=' < exp >

< estado-origem > ::= < nome-estado >

< estado-destino > ::= < nome-estado >

< nome-estado > ::= < identificador >

< evento-primitivo > ::= < identificador >

< condição-primitiva > ::= < identificador >

< variável > ::= < identificador >

Símbolos Terminais:

< true > ::= ([t T] [r R] [u U] [e E] | tr)

< false > ::= ([f F] [a A] [l L] [s S] [e E] | fs)

< changed > ::= (changed | ch)

< identificador > ::= [a - z A - Z] [a - z A - Z 0 - 9] *

< número > ::= [0 - 9] +