ATIFS: um Ambiente de Testes baseado em Injeção de Falhas por Software

Eliane Martins
eliane@dcc.unicamp.br

Relatório Técnico DCC-95-24

Dezembro de 1995

ATIFS: um Ambiente de Testes baseado em Injeção de Falhas por Software

Eliane Martins

DCC - UNICAMP
Cid. Univ.Zeferino Vaz CP 6065
13087-970 Campinas - SP
e-mail: eliane@dcc.unicamp.br

RESUMO

Uma das dificuldades no desenvolvimento de sistemas tolerantes a falhas diz respeito à sua validação. Essa dificuldade vem do fato de que os testes destes sistemas requerem a consideração de situações anormais, que ativem os mecanismos de tratamento de erros e de falhas existentes em tais sistemas.

Uma técnica que vem sendo muito utilizada para este fim é a injeção de falhas, que consiste em introduzir erros/falhas em um sistema, de maneira controlada, e observar o seu comportamento.

Neste texto é apresentado o **ATIFS**, um **A**mbiente para o **T**este baseado em **I**njeção de **F**alhas por **S**oftware, que provê suporte para o desenvolvimento e execução de testes de sistemas tolerantes a falhas, bem como para a análise dos resultados obtidos nos testes. Uma importante característica do ATIFS é que ele permite o uso de um modelo formal do sistema, o qual pode servir de base tanto para a geração automatizada dos testes a serem aplicados quanto para a obtenção de uma referência a ser empregada na análise dos resultados. Para a escolha das entradas de teste, o ATIFS oferece ao usuário a possibilidade de definir testes estatísticos, baseados na distribuição de probabilidades associada ao domínio de entrada. A realização de testes estatísticos baseados no modelo do sistema permite que o ATIFS forneça suporte tanto para a verificação do comportamento do sistema em presença de falhas, quanto para a avaliação de medidas da eficiência dos seus mecanismos de detecção/recuperação de erros/falhas, característica que o diferencia da maioria dos ambientes de testes por injeção de falhas.

PALAVRAS-CHAVE

injeção de falhas - testes estatísticos - testes formais - ambiente integrado de testes

ÍNDICE

I. INTRODUÇÃO

Sistemas computacionais são constituídos de uma série de componentes de hardware, de software e outros, que podem falhar eventualmente. Estas falhas podem levar o sistema a apresentar um defeito[1], ou seja, o serviço fornecido pelo mesmo não está de acordo com o que foi especificado [Laprie 92].

Para alguns sistemas, a ocorrência de defeitos pode implicar em custos muito elevados em termos econômicos ou até mesmo em termos de vidas humanas. Como as falhas não podem ser de todo evitadas, pode-se tentar contrabalançar o seu efeito através de redundância. Esta redundância pode ser aplicada para se tolerar falhas que ocorram em componentes de hardware, de software e até no sistema [Heimerdinger 92]. Assim, por exemplo, uma forma de compensar o efeito de falhas de hardware consiste em usar circuitos extras, como processadores duplicados ou memórias adicionais. A redundância é portanto a base das técnicas de **tolerância a falhas**, as quais visam garantir que o sistema continue a fornecer o serviço conforme à especificação mesmo em presença de falhas em alguns de seus componentes [Laprie 92].

O uso de técnicas que permitam evitar e/ou tolerar falhas por si só não é suficiente para garantir a confiança no serviço fornecido pelo sistema. Os métodos para a construção de sistemas não estão livres de falhas; portanto é necessário o uso de técnicas que permitam eliminar ao máximo as falhas residuais existentes no projeto e/ou implementação de sistemas. Estas técnicas, por sua vez, também são imperfeitas; não se podendo ainda eliminar de todo a possibilidade de ocorrência de falhas, é importante se fazer uma previsão do efeito das mesmas sobre o sistema. Estes dois aspectos dizem respeito à **validação** de um sistema, o que incluiria então [Laprie 92]:

- a **eliminação de falhas**, que envolve a verificação, o diagnóstico e a correção;
- a **previsão de falhas**, que visa obter, por avaliação, medidas que permitam caracterizar o comportamento do sistema em presença de falhas.

Um ponto crucial na validação de sistemas tolerantes a falhas diz respeito à validação dos seus mecanismos de tolerância a falhas (MTFs). A importância da validação desses mecanismos se deve a duas razões principais:

- a presença de falhas de projeto/implementação nesses mecanismos pode levar à deficiências no comportamento dos mesmos quando em presença de falhas para as quais eles foram projetados para tratar; essas deficiências podem levar o sistema a não mais fornecer o serviço correto (i.e, conforme à especificação);
- o efeito da eficiência dos mecanismos de tolerância a falhas sobre medidas tais como a confiabilidade do sistema (ver por exemplo os estudos apresentados em [Bouricius 69] e [Arnold 73]).

---

[1] Uma falha ("fault") é a causa direta de um erro. Um erro é a parte do estado do sistema que pode levar a um defeito ("failure"). Não existe ainda um consenso quanto à terminologia a ser usada em português; os termos usados aqui estão baseados em [Leite 87].

O uso de técnicas formais de verificação (como prova de programas), é altamente desejável para garantir a corretude do sistema, em particular, de seus MTFs. O uso de métodos analíticos (modelos de Markov, ...) também é necessário para a obtenção de medidas da eficiência desses mecanismos. No entanto, devido à complexidade dos sistemas tolerantes a falhas, sua aplicação é limitada a partes do sistema e à consideração de um número reduzido de falhas.

Estas técnicas devem então ser complementadas com a validação experimental. A **injeção de falhas** é uma técnica que vem sendo muito utilizada na validação de mecanismos de tolerância a falhas de um sistema pois permite validá-los em presença de entradas especiais, para as quais eles foram construídos para tratar: **as falhas**.

A injeção de falhas pode ser aplicada de diversas formas, em diferentes fases do ciclo de desenvolvimento de um sistema. Neste estudo, o enfoque é na sua aplicação como técnica de teste de uma implementação.

O teste por injeção de falhas permite que sejam cobertos os dois aspectos da validação mencionados acima [Arlat 89], onde se pode observar que:

- na **eliminação de falhas**, o objetivo é reduzir ao máximo as falhas de projeto/implementação existentes nos MTFs;
- na **previsão de falhas**, o objetivo é avaliar a eficiência dos MTFs. Neste caso os testes servem para se obter estimativas de parâmetros tais como o fator de cobertura e a latência de detecção de erros, que serão definidos mais adiante.

Na grande maioria dos estudos utilizando injeção de falhas os testes visavam particularmente a previsão de falhas [Kurlak 81, Crouzet 82, Lala 83, Segall 88b, Dilenno 91, Gunneflo 89, Czeck 90, Kanawati 92, Rosenberg 93, ...]. Decerto, os resultados obtidos serviram para revelar falhas de concepção existentes nos sistemas testados, mas nestes casos a eliminação de falhas era obtida como um subproduto, e não como um fim em si.

Por outro lado, os trabalhos que tratam a eliminação de falhas de maneira sistemática em geral não englobam a previsão de falhas [Winfrey 89, Avresky 91, Echtle 92].

Neste estudo será apresentada uma estratégia para a realização de testes por injeção de falhas. Esta estratégia é a base da definição de um ambiente, o ATIFS (Ambiente de Testes baseado em Injeção de Falhas por Software), que tem como principais características:

1. permitir a derivação automática dos testes a partir de um modelo formal do sistema, que descreva explicitamente o comportamento do mesmo em presença de falhas. Dessa forma é possível obter de forma integrada tanto as entradas funcionais, que vão ativar o sistema, quanto as falhas, que vão exercitar seus MTFs;
2. utilizar o modelo formal do sistema como referência, na eliminação de falhas, como forma de reduzir um problema comum aos testes, denominado *problema do oráculo*;

3. utilizar técnicas da teoria da amostragem que permitam não somente obter estimativas de boa qualidade, mas também determinar o erro cometido nestas estimativas;
4. dar suporte às atividades do processo de teste, o que compreende: a geração dos testes, a execução dos mesmos e a análise dos resultados.

Este texto está organizado da seguinte forma: na Seção II são apresentados os testes por injeção de falhas, suas formas de aplicação e seus atributos, seguido de uma apresentação dos principais aspectos da forma de injeção tratada neste estudo, que é a injeção de falhas implementada por software. A Seção III descreve a estratégia proposta, combinando a injeção de falhas com o teste formal (onde por teste formal entenda-se o teste baseado na existência de um modelo formal do sistema ou de seus MTFs). As Seções IV e V são relativas ao ATIFS: a primeira delas descreve as características do ambiente, e a outra descreve sua arquitetura. A Seção VI conclui o texto apresentando as direções para trabalhos futuros.

II. O TESTE POR INJEÇÃO DE FALHAS

Os aspectos gerais do método de injeção de falhas são introduzidos no item II.1. Trata-se de uma apresentação rápida, com o objetivo de situar o leitor no contexto deste estudo. Para os que desejem uma apresentação mais detalhada dos aspectos básicos, a referência [Martins 93] pode ser consultada; e para uma visão mais aprofundada do assunto, recomenda-se a leitura de [Arlat 92].

O item II.2 trata especificamente da injeção de falhas por software, a forma de injeção que é objeto deste estudo.

II.1 Apresentação

A **injeção de falhas**, nome genérico dado a um conjunto de técnicas utilizadas para acelerar a ocorrência de falhas, erros e defeitos em um sistema, vem sendo largamente utilizada na validação dos mecanismos de tolerância a falhas de um sistema, pois permite validá-los em presença de entradas particulares para as quais eles foram desenvolvidos para tratar: **as falhas**.

Na eliminação de falhas, a injeção de falhas tem por objetivo verificar se os MTFs se comportam de acordo com a sua especificação. Na previsão de falhas o objetivo é obter medidas da eficiência desses MTFs.

II.1.1 Formas de Aplicação

As falhas que afetam a confiabilidade de um sistema podem ser divididas em: *falhas de hardware* ("imitam" a consequência de falhas de hardware sobre o sistema) e *falhas de software* (falhas de projeto/implementação do software, designadas também como falhas de concepção).

A maioria dos estudos utilizando injeção de falhas são relativos à introdução de falhas de hardware, sendo também este o enfoque neste estudo.

As falhas de hardware podem ser aplicadas de diferentes formas, dependendo do nível de abstração utilizado para representar o sistema (modelo ou protótipo) e do nível de aplicação das falhas (físico ou lógico). As formas de injeção mais empregadas são a simulação de falhas, a injeção física de falhas e a injeção de falhas por software, apresentadas a seguir.

Na **simulação de falhas**, falhas lógicas são introduzidas em um modelo do sistema. Este modelo pode representar o comportamento ou a estrutura (em termos de portas e/ou transistores) do sistema. As falhas são injetadas no modelo com o intuito de analisar o processo de ativação de falhas e de propagação de erros, para avaliar o comportamento dos MTFs. As falhas podem ser aplicadas por módulos especialmente construídos para este fim [Czeck 90, Goswami 91, ...], ou através de mutações do modelo original [Jenn 92].

A **injeção física de falhas** consiste em aplicar as falhas físicas a um protótipo do hardware (e/ou do software) do sistema. As falhas são injetadas geralmente sobre os pinos de circuitos integrados [Kurlak 81, Crouzet 82, Lala 83, Damm 88, Arlat 90b, Madeira 93], mas pode-se também submeter os componentes ao bombardeamento por íons pesados [Gunneflo 89] ou ainda à aplicação de radiações eletromagnéticas [Kopka 88]. Neste caso os testes visam principalmente estudar o comportamento de MTFs implementados por hardware, mas pode-se também testar software tolerante a falhas [Martins 89]. As falhas são aplicadas por um dispositivo especialmente construído para este fim, que interage com o sistema em teste.

Na **injeção de falhas implementada por software**, falhas lógicas são introduzidas em um protótipo do software (e/ou do hardware) do sistema, com o objetivo de emular a consequência de falhas físicas. Esta técnica apresenta as seguintes vantagens, em relação às outras duas técnicas apresentadas:

- não necessita de equipamento especial de hardware, como é o caso da injeção física de falhas,
- pode ser aplicada mais cedo na fase de testes do que a injeção física, pois não é necessário que o hardware onde vai residir o sistema alvo esteja disponível;
- oferece maior facilidade no controle e observação do sistema durante os testes, como a simulação, sem apresentar o tempo elevado de processamento desta última.

Esta técnica é adequada para a validação de mecanismos de tolerância a falhas implementados por software, por ter a capacidade de injetar condições de erro específicas que permitam ativar esses mecanismos, o que não pode ser garantido na injeção física de falhas.

A injeção de falhas implementada por software, referenciada aqui simplesmente como *injeção por software*, é a técnica abordada neste estudo, e será apresentada com mais de detalhes no item II.2.

II.1.2 Caracterização

A injeção de falhas é caracterizada por um domínio de entrada e um domínio de saída. O *domínio de entrada* corresponde a um conjunto de falhas a injetar **F** e um conjunto de ativações **A** que especifica as entradas destinadas a exercitar o sistema e, em consequência, ativar as falhas injetadas. O *domínio de saída* é caracterizado por um conjunto **R** de dados coletados e por um conjunto de medidas **M** derivadas da análise e tratamento dos conjuntos **F**, **A** e **R**. Os conjuntos **FARM** constituem então os principais atributos da injeção de falhas.

A sequência de testes por injeção de falhas é composta por uma série de **experimentos**, cada qual sendo caracterizado por um ponto no espaço **F×A×R**. O número de experimentos a serem realizados depende do objetivo da validação (eliminação ou previsão de falhas), como mostra a Tabela II.1.

Tabela II.1. Os atributos **FARM** e os objetivos da validação da tolerância a falhas.

	eliminação de falhas	**previsão de falhas**
objetivo	eliminar o maior número possível de falhas de projeto/implementação do sistema.	obter medidas da eficiência dos mecanismos de tolerância a falhas do sistema.
F	pequeno número de falhas específicas, determinadas a partir das hipóteses feitas na fase de projeto do sistema.	número elevado de falhas, representativas das falhas que possam ocorrer em fase operacional.
A	entradas destinadas a exercitar as funções do sistema e ativar as falhas injetadas.	entradas representativas do perfil operacional do sistema.
R	indicações de ocorrência de eventos e/ou medidas de tempo associadas + informações para ajuda ao diagnóstico ("dumps" de memória, ...).	número de ocorrências de eventos e medidas de tempo associadas.
M	veredito: valor indicando se o sistema se comporta como o esperado.	estatísticas sobre a ocorrência de eventos e dos tempos entre as ocorrências.

II.2 A injeção por software

Nos itens a seguir serão apresentados os principais trabalhos utilizando injeção por software. Para sintetizar a apresentação, os trabalhos considerados foram analisados segundo os seguintes aspectos:

- o sistema alvo,
- os objetivos da validação,
- o tipo de falhas injetadas,
- as formas de injeção,
- a geração das falhas,
- o tamanho da seqüência de testes.

A Tabela II.2 resume as informações mencionadas para cada um dos trabalhos estudados. As duas primeiras colunas da tabela apresentam a referência principal, o nome do organismo, seguido do nome da ferramenta (em itálico), se for o caso. As outras colunas são relativas aos aspectos mencionados acima, os quais são apresentados resumidamente nos itens que se seguem.

II.2.1 O sistema alvo

Neste caso é indicado se o sistema em teste é uma arquitetura distribuída (**AD**), como em [Segall 88a, Dilenno 91, Lovric 93, Rosenberg 93]; ou uma aplicação tolerante a falhas (**APL**), como em [Kanawati 92] e [Kao 93], que testam aplicações rodando sob o Unix; ou mecanismos de tolerância a falhas específicos (**MTF**), como em [Winfrey 89], que trata do teste de mecanismos de detecção de erros em aplicações baseadas em troca de mensagens.

Tabela II.2. Algumas ferramentas e resultados experimentais obtidos na injeção de falhas por software.

Referência	Organismo (*ferramenta*)	Sist. Alvo	Objetivos	Tipo de Falha	Forma de injeção	Tipo de Teste	Tamanho da seq. de testes
[Segall 88b]	Un. Carn. Mell., IBM, System/Tech. Dev. Corp. (*FIAT*)	AD	A	FM	AFC	E ei	≈ 130.000
[Winfrey 89]	Un. de Columbia	MTF	V	FC	AFC	D, ef	—
[Avresky 91]	Lab. d'Autom. et d'An. Syst.	APL	V	—	—	D, ef	—
[Dilenno 91]	IBM, System/Tech. Dev. Corp.	AD	AV	FM, FP, FC	—	D, mf E, —	300 (testes determinísticos)
[Echtle 92]	Un. de Dortmund (*EFA*)	APL	V	FC	AFC	E/D, st	—
[Kanawati 92]	Un. do Texas at Austin (*FERRARI*)	APL	AV	FM, FP, …	AC	E, —	≈ 600.000
[Rosenberg 93]	Un. de Michigan (*SFI*)	MTF	AV	FM, FP, FC	A, AFC, AC	E, —	≈ 10.000
[Lovric 93]	Un. de Dortmund (*ProFI*)	AD	AV	FP	AC	—, —	≈ 90.000
[Kao 93]	Un. de Illinois at Urb.-Champ. (*FINE*)	APL	A	FM, FP, outras (1)	A, AFC, AC	E/D, —	≈ 500

Observações:
- "—": não fornecido
- "(1)": além de falhas de hardware, a ferramenta é capaz também de injetar falhas de software.

II.2.2 Os objetivos da validação

Neste caso é indicado se a validação teve por objetivos a verificação (**V**), a avaliação (**A**) ou ambos (**AV**).

Na maioria dos estudos, o objetivo visado é a avaliação de parâmetros tais como o fator de cobertura e a latência. Esses parâmetros podem ser usados tanto na construção de modelos analíticos para o cálculo da confiabilidade ou do desempenho, como em [Kao 93], quanto na validação de modelos pré-existentes, como em [Rosenberg 93]. Nestes estudos a verificação também pode ser considerada, mas como aspecto secundário.

Os estudos que abordam unicamente a verificação, por outro lado, visam principalmente exercitar o modelo do sistema, e não consideram o aspecto avaliação, como em [Avresky 91] ou [Echtle 92].

II.2.3 Tipos de falhas consideradas

Todas as ferramentas consideradas neste estudo são voltadas para a injeção de falhas de hardware[2],com exceção de FINE [Kao 93], que também leva em conta as falhas de software.

Os tipos de falhas considerados mais comumente para injeção seriam: *falhas de memória* (**FM**: alterações no conteúdo da memória, podendo ocorrer tanto no segmento de programa quanto no de dados); *falhas de processador* (**FP**: alterações no conteúdo de registradores, ou em bits de instruções, ou no endereço de destino de uma instrução de desvio, ou ainda em temporizadores); *falhas de barramento* (**FB**: alterações tanto nas linhas de endereçamento quanto nas linhas de dados, podendo afetar bits em instruções ou nos dados transmitidos através do barramento); *falhas de comunicação* (**FC**: alteração, perda, retardo ou duplicação de mensagens que circulam nos meios de comunicação).

II.2.4 Geração das falhas

Como em outras técnicas de teste, a determinação das falhas a serem injetadas envolve dois aspectos [Laprie 92]: estabelecimento de um critério de seleção e geração das falhas segundo o critério adotado. O critério de seleção pode ser estabelecido com base:

- nos modos de falha do sistema (**mf**): [Dilenno 91], apresenta uma análise dos modos de falha do sistema, dos seus efeitos e da sua criticidade (FMECA, de "Failure Modes, Effects and Criticality Analysis"), realizada nas várias fases do ciclo de desenvolvimento, a fim de identificar os modos de falha que tenham maior probabilidade de causar defeitos graves no sistema. Esses modos de falhas são induzidos através de injeção de falhas;
- na estrutura do programa (**st**): em [Echtle 92], as falhas são escolhidas de forma a cobrir ao máximo o código do programa em teste;
- na especificação do sistema (**ef**): em [Winfrey 89] e [Avresky 91] as falhas são escolhidas de forma a cobrir ao máximo o modelo funcional do sistema. Os testes também podem ser baseados em uma especificação informal (**ei**) do sistema, o que é comum quando o objetivo visado é a avaliação [Segall 88b].

De acordo com o critério adotado, a geração das falhas pode ser **determinística** (**D**), na qual as falhas a serem injetadas são determinadas a partir de uma escolha seletiva; ou **estatística** (**E**), na qual as falhas são selecionadas de acordo com uma distribuição de probabilidades associada ao domínio de entrada.

II.2.5 Formas de injeção

Tendo visto a grande variedade de sistemas existentes, a acessibilidade dos diversos componentes nos quais se deseja injetar falhas varia consideravelmente. Por essa

[2] Em vez de falhas de hardware deveria ser empregado erros de software, de acordo com as definições de falhas e erros [Laprie 92]. No entanto elas são tratadas aqui como falhas de hardware para significar que a ativação das mesmas é a causa de erros no software, seja no código, seja nos dados.

razão, as formas empregadas para injeção também variam; três formas de injeção podem ser identificadas [Rosenberg 93]:

- *injeção ativa* (**A**), realizada por um processo que é executado concorrentemente com a aplicação. Essa forma pode ser empregada para injetar falhas de memória em ambientes onde não haja proteção de acesso para outros processos;
- *alteração no fluxo de controle* (**AFC**) usada para alterar o comportamento do sistema: quando em modo injeção, o programa realiza uma seqüência de instruções alternativas, fazendo com que uma determinada função seja realizada de maneira incorreta. Esse modo é ideal para injetar falhas de comunicação ou para injetar falhas no sistema operacional, alterando-se os serviços que são oferecidos à aplicação em teste;
- *alteração de código* (**AC**) usada para acessar áreas da aplicação em teste que não são accessíveis por outros processos. Neste caso, o código da aplicação é alterado, fazendo com que operações incorretas sejam realizadas. Esta forma pode ser usada para injetar falhas de processador, por exemplo, onde as instruções incorretas "simulariam" falhas em unidades funcionais. Essa alteração pode ser feita em tempo de compilação [Rosenberg 93] ou durante a execução, como em [Kanawati 93]: quando a execução atinge um endereço especificado, um "alçapão" captura a instrução, altera-a e em seguida a instrução modificada é executada.

II.2.6 Tamanho da seqüência de testes

O tamanho da seqüência de testes, i.e., o número de experimentos a serem executados, é geralmente elevado, em parte devido às possibilidades de variação dos atributos que podem ser usados para definir as falhas, quais sejam [Arlat 90a]:

- *multiplicidade*: uma falha pode atingir um ou mais bits de uma dada posição de memória;
- *localização*: endereço de memória, ou parte da mensagem, onde a falha vai ser injetada;
- *modelo*: o modelo de falha comumente utilizado é do tipo bit preso em 0/1, pois estes fornecem uma boa cobertura das falhas de hardware [Lovric 93];
- *persistência temporal*: as falhas podem ser injetadas de maneira permanente ou temporária (transiente ou intermitente).

Além da variação dos atributos, é também importante variar o modo de ativação (c.f. II.1), devido a sua influência sobre o comportamento dos MTFs.

A título de exemplo, tome-se o estudo apresentado em [Segall 88b], no qual foram realizados 131.320 testes, correspondentes à injeção de falhas de memória. Para este tipo de falha, foram consideradas 3 instâncias distintas, cada qual correspondendo aos seguintes modelos: inversão de 2 bits em uma palavra de 32 bits; atribuição de valor 0 a um byte de uma palavra de 32 bits; atribuição de valor 1 a um byte de uma palavra de 32 bits. Quanto à persistência temporal, foram consideradas falhas pseudo-permanentes: as posições de memória eram alteradas ao se iniciarem os testes, e permaneciam como tal até o término dos

mesmos. Para os modos de ativação foram usados dois programas: um de multiplicação de matrizes e outro de ordenação. Para cada programa foram considerados diferentes tamanhos dos dados manipulados. A seqüência de testes foi dividida em 13 classes, correspondentes às combinações de interesse entre os modos de ativação e as falhas. Testes preliminares foram realizados com o objetivo de determinar o tamanho de cada classe, bem como os atributos das falhas a serem variados durante os testes.

II.2.7 Comentários finais

Com base na tabela II.2 pôde-se constatar que na maioria dos casos a validação tem por objetivo a previsão de falhas. Naturalmente que os resultados obtidos também servem para a eliminação de falhas, como tem sido mostrado em diversos trabalhos (por exemplo, aqueles indicados "AV" tabela); mas nestes casos a verificação é considerada de forma "ad hoc", não se utilizando nenhuma abordagem sistemática para isso.

Os trabalhos que abordam a verificação de maneira sistemática baseiam-se na existência de um modelo que caracterize o sistema ou seus MTFs. Em [Dilenno 91] tem-se um modelo de falhas do sistema; em [Winfrey 89] e [Avresky 91], são utilizados modelos do comportamento do sistema. Echtle et al. utilizam um grafo do fluxo controle, o qual serve de base para os testes estruturais do programa [Echtle 92].

No entanto, estes estudos baseiam-se na realização de testes determinísticos, pois visam primordialmente obter a melhor cobertura possível do modelo do sistema (ou de seus MTF) segundo o critério de seleção adotado. Desta maneira, o segundo objetivo da validação, que é a avaliação, não é abordado.

Uma outra dificuldade na injeção de falhas é determinar se as saídas produzidas pelo sistema estão corretas ou não; é o denominado *problema do oráculo* [Weyuker 82]. Este problema é comum a qualquer técnica de verificação dinâmica (teste), e neste caso, não existe uma solução genérica.

Na maioria dos estudos utilizando-se injeção de falhas a solução empregada consiste em comparar os resultados obtidos após as injeções com aqueles obtidos quando da execução sem falhas da aplicação, a qual serve portanto como referência [Segall 88b, Echtle 92, Lovric 93, Rosenberg 93]. Este esquema é denominado de teste por comparação [Pressman 92].

Quando se dispõe de uma especificação formal do sistema, a análise dos resultados pode ser feita com relação a essa especificação. Até o momento, poucos trabalhos em injeção de falhas têm se utilizado de uma especificação formal para auxiliar na verificação. Um pioneiro nessa área é o estudo mostrado em [Avresky 91], onde um modelo formal, representando o comportamento dos MTFs, é executado em paralelo com a aplicação em teste, servindo portanto como referência para a análise dos resultados.

Na próxima seção é apresentada a estratégia proposta neste estudo que permite tratar de maneira sistemática os dois aspectos da validação.

III. COMBINANDO INJEÇÃO DE FALHAS E TESTE FORMAL

III.1 Princípios

A abordagem proposta nesta estudo baseia-se na combinação de teste formal com o teste por injeção por software. O termo *teste formal* é usado aqui como uma forma abreviada de designar o teste funcional a partir de uma especificação formal representando os MTFs em teste.

O uso de uma especificação formal nos testes por injeção por software apresenta as seguintes vantagens:

- maior facilidade para a geração dos testes, pois a existência de uma especificação formal dos MTFs serve de base tanto para a obtenção das entradas de teste (<f, a>), quanto para monitorar a cobertura dos testes obtidos com relação a essa especificação;
- redução do problema do oráculo pois a especificação formal pode ser usada como referência;
- facilidade de diagnóstico e correção de erros.

A abordagem apresentada aqui tem por objetivo cobrir os dois aspectos da validação, quais sejam a eliminação de falhas e a previsão de falhas. Neste texto os principais pontos considerados na abordagem podem ser resumidos pelas seguintes questões:

i. como gerar os testes, usando a especificação formal, de maneira a cobrir estes dois aspectos?

ii. na eliminação de falhas: como determinar o número adequado de testes a realizar? como determinar se os resultados observados estão corretos?

iii. na previsão de falhas: como determinar o número adequado de testes a realizar, de forma a obter boas estimativas dos parâmetros desejados? como obter estas estimativas?

Os itens a seguir apresentam mais detalhes da estratégia de testes da tolerância a falhas, baseadas nos aspectos mencionados acima. Primeiramente é apresentado, a título de exemplo, um modelo genérico do comportamento de sistemas em presença de falhas e sua relação com os atributos FARM, que caracterizam os testes por injeção de falhas.

III.2 Modelo do comportamento de um sistema em presença de falhas

III.2.1 Descrição do modelo

Neste item é apresentado um modelo formal de sistema baseado em Máquina de Estados Finitos (MEF). Este modelo, descrito em [Arlat 90a], representa o comportamento de um sistema em presença de falhas e é na verdade baseado em uma MEF estendida, que leva em conta explicitamente as falhas. O modelo é então dado pela sêxtupla (**D**, **Y**, **F**, **Z**, **U**, *f*), onde:

D conjunto de entradas externas; d(t) ∈ **D** é um vetor das entradas presentes no sistema no instante t;

Y conjunto de estados correntes; y(t) ∈ **Y** é um vetor dos estados correntes no instante t;

F conjunto de falhas[3]; f(t) ∈ **F** é um vetor das falhas presentes no sistema no instante t. **F** inclui também o vetor f_0, representando a ausência de falhas;

Z conjunto de estados finais; z(t) ∈ **Z** é um vetor representando os novos estados do sistema após o decorrer de um intervalo δt, caracterizando o tempo necessário para a estabilização do estado interno ou o término do período de amostragem. Assim, z(t)=y(t+δt);

U conjunto de saídas do sistema, representando o serviço por ele prestado; o vetor u(t) ∈ **U** representa as saídas emitidas pelo sistema no instante t;

f corresponde à função desempenhada pelo sistema: **D** × **Y** × **F** → **Z** × **U**.

O funcionamento do sistema na ausência de falhas pode ser escrito como:

$$f[d(t), y(t), f_0(t)] = [z(t), u(t)] \quad \text{(III.1)}$$

A utilização da variável *t* permite levar em conta não somente os erros de valor mas também os erros de tempo. A fim de simplificar a escrita, será usada daqui para frente a notação: (d, y, f_0; t) = (z, u; t).

Dado em (III.1) a expressão do comportamento correto do sistema, podemos então definir o impacto de uma falha no sistema como:

$$\forall\ t, \exists\ d(t) \text{ e/ou } y(t) \mid (d, y, f; t) \neq (d, y, f_0; t) \quad \text{(III.2)}$$

onde f(t) ≠ f_0.

A expressão (IIII.2) descreve portanto o comportamento do sistema em presença de uma falha ativa, i.e., que deu origem a um erro. Esse erro caracteriza-se pela alteração observável do estado do sistema, ou seja:

$$(d, y, f; t) = (z', u; t) \neq (z, u; t) \quad \text{(III.3)}$$

O erro gerado em consequência da falha ativa no sistema pode ocasionar um defeito, i.e., o sistema não fornece o serviço apropriado (conforme à especificação). Essa situação pode ser caracterizada de duas formas, dependendo se o estado final do sistema é alterado ou não:

$$(d, y, f; t) = (z', u'; t) \neq (z, u; t) \quad \text{(III.4a)}$$

$$(d, y, f; t) = (z, u'; t) \neq (z, u; t) \quad \text{(III.4b)}$$

Em ambos os casos vemos que a saída fornecida, u'(t), é diferente da esperada, u(t), caracterizando assim o fornecimento do serviço inapropriado.

Diversas situações de erro, bem como de defeito, podem ser representadas por esse modelo, como pode ser visto em [Arlat 90a]. Para mostrar o efeito dos mecanismos de tolerância a falhas (MTFs) no sistema, é necessário introduzir dois conceitos: estado dinâmico e estado estático. O *estado dinâmico* é caracterizado pelas variáveis internas das

[3] Consideram-se aqui unicamente as falhas conformes à hipótese de falhas do sistema.

quais depende a evolução do sistema no instante considerado. O *estado estático* é caracterizado pelas variáveis internas que não são sensibilizadas no instante considerado.

O comportamento dos MTFs em presença de falhas pode ser descrito, em termos destes conceitos, como se segue:

a. **detecção de um erro**: essa detecção pode estar baseada na

- observação de uma alteração no estado dinâmico:

$$\forall\ f(t), \exists\ (d, y; t) \mid (d, y, f; t) \neq (z_s, z'_d, u; t) \quad \text{(III.5)}$$

onde z_s estado estático esperado,

z'_d estado dinâmico alterado.

- sensibilização de um estado estático e conseqüente alteração no estado dinâmico:

$$\forall\ y'_s, \exists\ (d, y_d; t)\ \ (d, y'_s, y_d, f_0; t) = (z'_s, z'_d, u; t) \quad \text{(III.6)}$$

b. **recuperação de erro**: consiste em restaurar o comportamento interno do sistema

$$f\left(d, y_s, y_d^{\copyright}, f_0; t\right) = \begin{cases} (z_s, z_d, u; t) \\ (z_s^{\copyright}, z_d, u; t) \end{cases} \quad \text{(III.7)}$$

de forma a que o erro não seja recorrente:

$$\forall t^{\copyright} \geq t, f(d, y_s^{\copyright}, y_d, f_0; t) = \begin{cases} (z_s, z_d, u, t^{\copyright}) \\ (z_s^{\copyright}, z_d, u; t^{\copyright}) \end{cases} \quad \text{(III.8)}$$

c. **mascaramento do erro**: o sistema deve continuar a fornecer o serviço correto,

$$f(d, y, f; t) = \begin{cases} (z, u; t) \\ (z^{\copyright}, u; t) \end{cases} \quad \text{(III.9)}$$

mesmo em presença de erro

$$f(d, y^{\copyright}, f; t) = \begin{cases} (z, u; t) \\ (z^{\copyright}, u; t) \end{cases} \quad \text{(III.10)}$$

III.2.2 Relação com os atributos FARM

A figura III.1, adaptada de [Arlat 90a] mostra a relação entre os atributos FARM e os conjuntos definidos no item anterior.

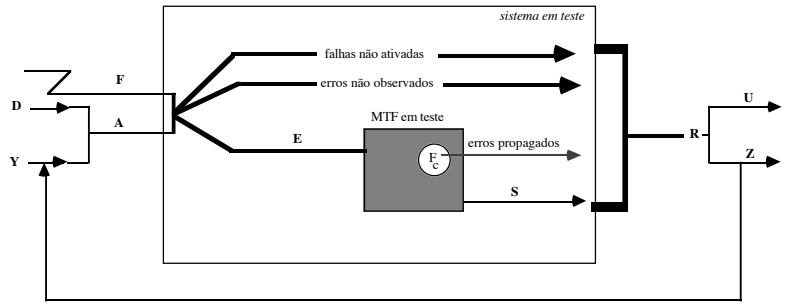

Fig. III.1. Caracterização do comportamento em presença de falhas.

A figura mostra um sistema tolerante a falhas com detalhes do MTF em teste. Dessa figura vemos que as informações relevantes para o teste são:

- o conjunto **E**, de erros injetados, caracterizado pelos erros que se propagam até o MTF em teste,
- o conjunto **S**, síndrome de erro, que caracteriza os sinais de detecção, alarmes ou mensagens de erro emitidos como parte da ação do MTF.

O conjunto **E**, que ativa o MTF, é obtido por uma combinação dos conjuntos que constituem o domínio de entrada, quais sejam:

- o conjunto **F**, das falhas injetadas,
- o conjunto **A**, os modos de ativação do sistema, constituído pelos conjuntos **D** e **Y**, apresentados no item anterior.

O conjunto **S** pode conter elementos dos dois conjuntos de saída apresentados no item anterior:

- o conjunto **Z**, definindo o estado em que se encontra o sistema,
- o conjunto **U**, caracterizando o serviço fornecido pelo sistema.

Desse modo o conjunto **R**, de resultados coletados, deve incluir elementos tanto de **Z** quanto de **U**.

O conjunto **M** caracteriza um conjunto de medidas binárias (veredito de conformidade) ou estatísticas (cobertura, ...) obtidas a partir da análise de **F**, **A** e **R**.

A figura III.1 mostra também que em caso de falhas de concepção - $\mathbf{F}_c$ - do MTF, o erro injetado pode se propagar e provocar um defeito no sistema. Também são mostrados casos de experiências não significativas com relação aos MTFs testados, que correspondem, por um lado, às falhas não ativadas durante o decorrer da experiência, e por outro lado, aos erros que não foram observados pelo MTF em teste. Um exemplo do primeiro caso seria a injeção de falhas de memória afetando bits em posições que não são solicitadas no decorrer da execução. Já no segundo caso, os erros podem não ser observados por diferentes razões:

- os erros podem ter sido apagados, sendo a sua propagação interrompida, caso dados corretos venham substituir os dados incorretos, antes que estes sejam utilizados;
- os erros podem ter sido detectados (e eventualmente tratados) por outros MTFs que não aquele que é submetido aos testes; é o *efeito de dominação*, identificado em [Arlat 91];
- os erros permanecem latentes até o fim do experimento, o que pode ocorrer devido à limitada duração do mesmo.

III.3 Escolha das entradas de teste

Uma entrada consiste no par <f, a>. A determinação desses pares depende dos objetivos da validação, como foi visto em II.1:

a. na eliminação de falhas é revelar o maior número possível de falhas de concepção, i.e., cobrir o máximo possível de elementos de $\mathbf{F}_c$. Devido à inexistência de um modelo de falhas de concepção amplamente reconhecido, deve-se então procurar exercitar o MTF em teste ao máximo; para isso as entradas devem corresponder à aplicação efetiva de um elemento de **E**;

b. na previsão de falhas os testes visam obter estimativas de medidas da eficiência do MTF. Para que as estimativas sejam representativas das medidas reais, os pares <f, a> devem representar situações de erro que o sistema deve encontrar em fase de operação.

De acordo com (a), pode-se deduzir que o ideal seria a realização de testes determinísticos (c.f. II): **F** e **A** seriam escolhidos após uma análise detalhada do MTF em teste, a fim de obter pares <f, a> que dêem origem a elementos de **E**.

Essa idéia é reforçada pelo fato de que é igualmente desejável que as entradas possam ser re-aplicadas, para controlar a eficácia das modificações eventuais resultantes das correções realizadas.

A análise detalhada do sistema para a realização dos testes determinísticos pode se tornar impraticável, sobretudo quando se consideram as falhas temporárias. No entanto, como foi visto na seção II, alguns estudos utilizam esta forma de geração de falhas. Uma outra limitação do teste determinístico é que eles não permitem a obtenção de medidas da eficiência dos MTFs com um bom nível de confiança.

Uma outra alternativa seria a realização de testes estatísticos. Estes testes vêm sendo muito utilizados para o hardware, e ultimamente vêm chamando a atenção da comunidade de software; estudos mostram que eles têm um grande potencial para a

validação [Thévenod 91a, Thévenod 91b, ...]: dependendo da distribuição associada ao domínio de entrada, servem tanto para eliminação quanto para a previsão de falhas. Uma das dificuldades dos testes estatísticos está em determinar a distribuição adequada aos objetivos da validação.

Em suma, ambos os tipos de teste apresentam suas vantagens e suas limitações. Afim de explorar melhor as potencialidades de cada um, pode-se adotar a estratégia proposta em [Thévenod 91a], que consiste em combiná-los da seguinte forma:

Passo 1. *Realização de testes estatísticos para a eliminação de falhas:*

As entradas de teste são escolhidas aleatoriamente, com uma probabilidade de entrada que permita reduzir o número de experimentos necessários para cobrir o critério de seleção adotado. Espera-se desse modo revelar um grande número de falhas de $\mathbf{F}_c$.

Passo 2. *Realização de testes determinísticos para a eliminação de falhas:*

Testes adicionais são realizados para selecionar as entradas que não puderam ser testadas no Passo 1, devido à probabilidade delas ser insuficiente para o número de experimentos realizados.

Passo 3. *Realização de testes estatísticos para a previsão de falhas:*

As entradas são escolhidas aleatoriamente, com uma probabilidade que reflita a distribuição encontrada durante a vida operacional (*perfil operacional*). Dessa forma, as medidas da eficiência dos MTFs podem ser consideradas como representativas dos parâmetros "reais". Neste passo ainda podem ser encontradas novas falhas de $\mathbf{F}_c$, as quais devem ser diagnosticadas e corrigidas.

Uma das dificuldades da estratégia proposta diz respeito à complexidade do modelo a ser tratado. Modelos de comportamento dinâmico são em geral complexos; à medida em que se requer maior confiabilidade dos sistemas, seus mecanismos de tolerância a falhas (MTF) vão se tornando mais sofisticados, o que só faz aumentar essa complexidade. Ao se levar em conta explicitamente as falhas, a construção de um modelo que represente fielmente o sistema pode ser irrealizável.

O que se costuma fazer na prática é dividir o conjunto de falhas consideradas, F ($F \supseteq \mathbf{F}$), em partições disjuntas, denominadas de classes (ou tipos) de falhas (c.f. III.2). Assim, por exemplo, o conjunto de falhas de memória pode ser dividido nas seguintes sub-classes: falhas afetando 1 bit; falhas afetando 2 bits; falhas afetando uma palavra, etc. Mesmo assim, se o modelo completo continuar intratável, pode-se também dividi-lo em partições, obtendo-se testes para cada partição em separado (ver por exemplo, [Katsuyama 91, Thévenod 92, Vuong 94]).

Uma outra dificuldade é relativa à determinação do número de testes a serem realizados. Este valor depende da distribuição associada ao domínio de entrada, a qual pode variar segundo os objetivos da validação. Os itens a seguir mostram esta dependência.

III.4 Eliminação de falhas

No que diz respeito à eliminação de falhas dois aspectos devem ser ressaltados: por um lado, a escolha da seqüência de testes e por outro, a determinação da validade ou não dos resultados observados (o problema do oráculo). Estes dois aspectos constituem duas das dificuldades maiores dos testes, sendo agravados quando se trata da realização de testes estatísticos; eles são tratados nos itens a seguir.

III.4.1 Determinação da Seqüência de Testes

Na eliminação de falhas os testes têm por objetivo exercitar ao máximo os MTFs. Como um teste exaustivo é geralmente impossível, um **critério de seleção** deve ser estabelecido para determinar um subconjunto de elementos do modelo (pois trata-se aqui de testes funcionais) que vão ser exercitados durante os testes. Esse subconjunto deve ser finito, para que seus elementos tenham a chance de ser executados por pelo menos uma entrada.

Seja C um critério de seleção e S_C o subconjunto de elementos que deverão ser exercitados durante os testes de acordo com esse critério. Por exemplo, no caso de máquinas de estados finitos (MEF)[4], pode-se selecionar os testes de forma que todas as transições sejam cobertas; neste caso:

C = "cada transição deve ser executada pelo menos uma vez"

S_C = {transições da MEF}

Conhecendo-se a probabilidade associada ao domínio de entrada pode-se obter, para cada elemento $k \in S_C$, a probabilidade dele ser executado ao menos 1 vez (p_k). Esta probabilidade é calculada em função da combinação das probabilidades das entradas que permitem que o elemento k seja executado.

O número de testes a serem executados, N, é estabelecido de tal forma que cada $k \in S_C$ tenha uma probabilidade no mínimo igual a Q_N de ser executado ao menos 1 vez durante a aplicação dos testes. O valor Q_N é denominado **qualidade do teste com relação ao critério** C. Este conceito, estabelecido em [Thévenod 91a] para os testes estatísticos, foi extendido aos testes determinísticos [Thévenod 92].

Nos testes determinísticos, as entradas de teste são escolhidas a priori, de modo que cada elemento de S_C seja executado pelo menos 1 vez[5]; tem-se então: $Q_N = 1$.

Nos testes estatísticos as entradas são escolhidas aleatoriamente; por isso não se pode garantir, dado um número finito de testes, que cada elemento de S_C seja executado pelo menos uma vez. Assim sendo, $Q_N < 1$. Portanto, para um Q_N pré-fixado, deve-se escolher N tal que:

---

[4] Estudos relativos à realização de testes estatísticos a partir de uma MEF podem ser vistos em [Sidhu 89] e [Thévenod 92]

[5] Geralmente essa escolha é feita de modo que cada elemento seja executado exatamente 1 vez, para reduzir o número de testes necessários.

$$\forall\, k \in S_C \quad p_k \geq Q_N \qquad \text{(III.11)}$$

A determinação de N não será apresentada aqui, mas pode ser vista nas referências supra-citadas. Cabe ressaltar a dependência de N na distribuição associada ao domínio de entrada, sendo portanto muito importante a escolha adequada dessa distribuição.

Em resumo, a realização de testes estatísticos visando a eliminação de falhas envolve dois aspectos principais:

i. escolha da distribuição de probabilidades associada ao domínio de entrada que vão ser usadas para determinar as probabilidades p_k de se exercitar cada elemento de S_C ;

ii. determinação do número de testes N que permita atingir uma dada qualidade de teste Q_N dadas as probabilidades p_k inferidas em (i).

III.4.2 Determinação de um oráculo

Um outro aspecto importante na eliminação de falhas é um problema conhecido no teste de software como o *problema do oráculo*: como determinar as saídas corretas que o programa deve produzir em resposta às entradas fornecidas? No caso de existir uma especificação formal do sistema, esse oráculo deve ser baseado nessa especificação. Existem diferentes soluções, conforme a análise seja integrada aos testes ou realizada em separado.

No caso de *análise integrada aos testes*, os testes gerados contêm não somente as entradas a serem aplicadas mas também as saídas esperadas. Neste caso, a existência da especificação formal é útil na geração dos testes, mas é desnecessária durante a execução dos mesmos.

Na *análise separada dos testes*, o comportamento observado é comparado à especificação. Duas formas de análise podem ser feitas nesse caso: análise em linha e análise do traço de execução.

Na *análise em linha*, pressupõe-se que existe uma representação executável da especificação, a qual é executada em paralelo com a implementação em teste. Essa forma é baseada nos conceitos do observador e do executante, introduzidos em [Ayache 79]. O *observador* é um modelo executável, geralmente uma versão simplificada da especificação, e o *executante* é a implementação em teste.

Na segunda forma, analisa-se o *traço de execução*, como é denominada a seqüência de eventos observados durante os testes. A partir da especificação formal é construído um *analisador de traço*, que irá examinar o traço de execução a fim de determinar se ele é válido ou não. A referência [Bochman 89] apresenta um estudo sobre essa forma de análise.

III.5 Previsão de falhas

Na previsão de falhas, o objetivo dos testes estatísticos é obter estimativas, com boa precisão, a partir dos dados observados.

As estimativas obtidas nos testes por injeção de falhas são relativas à eficiência dos MTFs. Um parâmetro importante usado para medir essa eficiência é o **fator de cobertura** (ou cobertura da tolerância a falhas) [Bouricius 69], o qual geralmente é definido em termos probabilísticos como:

$$\boldsymbol{c} = \Pr\{\text{o serviço correto é fornecido} \mid \text{uma falha é ativada}\}$$

Diversos estudos mostram a influência da cobertura da tolerância a falhas sobre medidas da segurança no funcionamento[6] [Bouricius 69, Arnold 73, ...]. Sua avaliação é, em consequência, um aspecto de suma importância na validação da tolerância a falhas.

Dois parâmetros importantes na avaliação da cobertura são a dormência de falhas e a latência de erros. A **dormência** de falhas é definida como o intervalo de tempo entre a ocorrência de uma falha e sua ativação na forma de erro. A **latência** de um erro é o intervalo de tempo entre a ativação da falha que o gerou e sua detecção pelos mecanismos de tolerância a falhas.

Os MTFs são construídos de acordo com as classes de falhas que o sistema deve tolerar, que constituem as **hipóteses de falhas**, e que são estabelecidas nas fases iniciais do desenvolvimento do sistema. Portanto, a cobertura que eles fornecem é fortemente ligada aos tipos de erros aos quais eles são confrontados.

A acumulação de falhas pode se manifestar na forma de erros complexos, formados pela combinação de erros em diferentes partes do sistema; vê-se assim que a cobertura e os parâmetros temporais (dormência e latência) estão fortemente relacionados. Com relação à dormência, tem-se que um erro não pode ser detectado enquanto uma falha não for ativada. A presença de falhas dormentes no sistema pode violar as hipóteses de falhas, levando a um defeito do sistema. No que diz respeito à latência, é importante que ela seja curta, para limitar a propagação de erros através do sistema. Uma latência muito grande pode levar ao defeito dos MTFs, dado que eles podem se confrontar com situações de erro que não foram levados em conta na sua concepção.

Outros parâmetros podem ser obtidos experimentalmente: a probabilidade de ocorrência de defeito dada a ativação de uma falha, o MTTF ("Mean Time To Failure"), etc. Qualquer que seja o parâmetro procurado, designado aqui genericamente como Θ, considera–se que seu valor será estimado a partir de uma amostra, i.e, a partir de um subconjunto dos elementos do domínio de entrada $\{\mathbf{F} \times \mathbf{A}\}$. Neste caso, dois aspectos devem ser considerados: como obter a estimativa desejada? como determinar a margem de erro cometida?

Com relação ao primeiro aspecto, deseja-se determinar uma estimativa de boa qualidade de Θ. Esta estimativa, designada aqui por θ, é uma função dos valores amostrados, sendo portanto uma variável aleatória. A qualidade de uma estimativa pode ser julgada com base nos seguintes pontos [Green 72, cap. 9]:

---

[6] A segurança no funcionamento de um sistema é a propriedade que permite a seus usuários ter confiança no serviço fornecido por esse sistema. Cumpre notar uma vez mais que a definição é referente ao termo "dependability" , da terminologia para a área de Tolerância a Falhas apresentada em [Laprie 92], sendo que o termo em português é baseado em [Leite 87].

- a estimativa deve ser *consistente*, i.e., a probabilidade de que θ se aproxime do valor procurado Θ tende para 1 quando o tamanho da amostra cresce indefinidamente;
- a estimativa deve ser *não tendenciosa*, i.e., o valor esperado de θ deve ser igual a Θ para todos os valores possíveis de θ;
- a variância de uma estimativa não tendenciosa deve tender a 0 à medida que o tamanho da amostra cresça indefinidamente[7].

Com relação ao segundo aspecto, além do valor de θ obtido sobre uma amostra, deseja-se determinar um *intervalo de confiança* em torno desse valor, indicando a ordem de grandeza do erro cometido na amostragem. Este intervalo indica a faixa de valores dentro da qual se encontra o valor real, Θ, para um determinado *coeficiente de confiança*. O coeficiente de confiança expressa a probabilidade de que o valor real esteja neste intervalo.

A realização de testes por amostragem aleatória requer a aplicação de métodos da teoria de amostragem, com o intuito de simplificar a obtenção de amostras, reduzindo assim o esforço e o tempo requeridos para os testes. A apresentação desses métodos foge ao escopo desse estudo. Um estudo sobre a aplicação de métodos de amostragem na avaliação da cobertura da tolerância a falhas pode ser vista em [Martins 92].

Em resumo, o teste por injeção de falhas visando a previsão de falhas envolve a realização dos seguintes passos:

i. escolha da distribuição de probabilidades associada ao domínio de entrada que reflita o perfil operacional do sistema, a fim de que as estimativas obtidas sejam representativas dos valores reais dos parâmetros procurados;

ii. determinação do número de testes N que permita atingir um dado coeficiente de confiança na precisão das estimativas obtidas a partir dos testes realizados segundo a distribuição de probabilidades estabelecida em (i).

IV. CARACTERÍSTICAS DO AMBIENTE

Esta seção descreve o ambiente de testes, **ATIFS** (**A**mbiente de **T**estes baseado em **I**njeção de **F**alhas por **S**oftware), que implementa a estratégia de testes descrita na seção anterior. O ATIFS visa primordialmente a realização de testes da tolerância a falhas, mas deve também dar suporte a outros tipos de teste, descritos no item IV.1. Os requisitos do ambiente são apresentados nos itens IV.2 e IV.3.

IV.1 Tipos de teste suportados

O principal objetivo do ATIFS é dar suporte à estratégia de teste por injeção de falhas descrita na seção anterior. No entanto, prevê-se também o suporte a outros tipos de teste, tais como testes de conformidade e testes de desempenho.

[7] A variância de uma variável aleatória mede a variabilidade de seu valor esperado. Portanto, se a variância for pequena, significa que a estimativa não tendenciosa está próxima do valor real do parâmetro estimado.

Os **testes de conformidade**, que têm por objetivo determinar se a implementação se comporta de acordo com a especificação.

Os **testes de desempenho** têm por objetivo determinar se o sistema satisfaz aos requisitos de desempenho. Segundo [Legrand90] estes testes podem ser sub-divididos em:

- *testes de capacidade*: permitem determinar o comportamento do sistema quando se tem um grande número de entidades de teste operando simultaneamente, e determinar a degradação do serviço fornecido em função do aumento do número de entidades;
- *testes de resistência*: permitem determinar, para uma carga constante, a degradação do serviço fornecido ao longo do tempo;
- *testes de carga*: visam observar o comportamento do sistema em presença de uma carga importante;
- *avaliação do desempenho*: visam obter medidas do desempenho do sistema, tais como vazão máxima e o tempo de reação a eventos diversos.

IV.2 Requisitos funcionais

As funcionalidades a serem oferecidas pelo ambiente visam cobrir todas as atividades do processo de teste, quais sejam:

- desenvolvimento dos testes,
- geração do script,
- execução dos testes,
- análise dos resultados.

Estas atividades constituem os principais subsistemas do ATIFS, como mostra a figura IV.1. Estes subsistemas estão apresentados com mais detalhes na seção V.

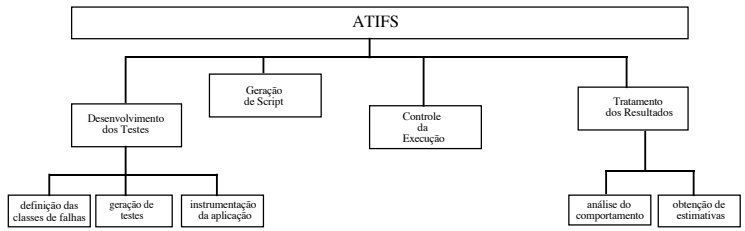

Fig. IV.1. Estrutura do ambiente de testes

IV.3 Requisitos não funcionais

Além das funcionalidades mencionadas, o desenvolvimento do ambiente deverá levar em conta as seguintes metas:

- Independência da implementação em teste

É necessário um investimento elevado tanto em tempo quanto em recursos humanos e materiais para se desenvolver tal tipo de ambiente. Por isso é interessante que o mesmo seja o mais independente possível da implementação em teste, de forma a ser aplicável ao teste de diferentes sistemas.

- Facilidade de adaptação a diferentes configurações de teste

  Como foi mencionado anteriormente, o ambiente a ser desenvolvido deverá permitir a realização de diferentes tipos de testes. Assim sendo, ele deve ser facilmente adaptável a diferentes configurações de teste, requerendo o mínimo de mudanças na sua estrutura.

- Facilidade de adaptação a diferentes sistemas

  Os sistemas abertos permitem a interconexão de sistemas heterogêneos (equipamentos de diversos portes e características, sistemas operacionais variados). Portanto, o ambiente deve ser utilizável em diferentes sistemas de comunicação, rodando sob diferentes sistemas operacionais, requerendo para tanto um mínimo de alterações.

- Convivialidade

  O ambiente deverá oferecer uma interface amigável ao usuário, permitindo a sua intervenção nas diferentes fases de teste.

- Facilidade de extensão

  O ambiente deverá ser expansível, i.e., o acréscimo de novas funcionalidades não deve requerer mudanças estruturais profundas.

- Robustez

  As falhas injetadas no sistema alvo n~ao devem afetar a continuidade dos serviços fornecidos pelo ambiente.

V. DESCRIÇÃO DO AMBIENTE

V.1 Arquitetura

O ambiente é composto de uma série de subsistemas (c.f. IV.2) que necessitam trabalhar de forma integrada. Para isso, a arquitetura do ambiente deve garantir os seguintes tipos de integração [Thomas 92]:

- *integração dos dados*, para permitir que os diferentes subsistemas possam compartilhar as informações geradas por eles, provendo os mecanismos básicos de armazenamento e gerenciamento da informação;
- *integração do controle*, para permitir que os diversos subsistemas possam ser combinados, de maneira flexível, de acordo com as preferências de cada projeto que utilize o ambiente;

- *integração da apresentação*, para proporcionar ao usuário uma forma homogênea tanto para a apresentação da informação quanto para as formas de interação com cada subsistema;
- *integração do processo*, para garantir que os sistemas possam interagir de maneira efetiva no suporte ao processo de teste.

Para atingir essa integração completa o ambiente deve conter, além dos subsistemas já mencionados, os seguintes componentes [Pressman 92, cap. 23]: um banco de dados, que vai servir como um depósito global, onde são armazenadas todas as informações; um subsistema para o controle de objetos, gerenciando a troca de informações; um mecanismo para controle dos subsistemas e uma interface com o usuário que permita o acesso consistente aos subsistemas que compõem o ambiente. Existem alguns modelos (ou "frameworks") para representar esses componentes; será considerado aqui um modelo em camadas, apresentado na referência supra-citada, e mostrado na figura V.1.

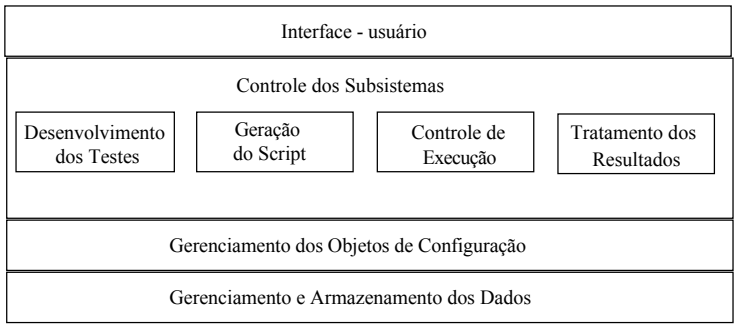

Fig. V.1. Arquitetura do ambiente

Os componentes dessa arquitetura estão descritos em detalhes nos itens a seguir.

V.2 Interface-usuário

O ambiente deve fornecer um único ponto de entrada para o usuário, através de uma interface amigável. Essa interface deverá oferecer ao usuário as facilidades disponíveis para o diálogo homem-máquina, o que inclui mecanismos de janelas, menus, uso do mouse, entre outras.

Além de fornecer ao usuário um meio fácil e eficiente de ter acesso aos diversos subsistemas, a interface deve ter ainda os requisitos abaixo:

- *uniformidade*, tanto na interação entre o usuário e os componentes do ATIFS, quanto na interação entre estes componentes e a interface;

- *independência* entre a interface e os demais componentes, de modo que estes possam evoluir durante o tempo de vida do ambiente sem maiores dificuldades.

V.3 Subsistemas

Esta camada compõe a parte variável da arquitetura, pois contém os subsistemas que implementam as funcionalidades de teste, subsistemas estes que podem evoluir ao longo do tempo, na medida em que novos procedimentos ou técnicas de teste sejam acrescentados ao ambiente. O componente de **Controle dos Subsistemas** é responsável pela comunicação e sincronização entre os subsistemas, bem como pela coordenação do acesso à informação armazenada no banco de dados, sendo portanto responsável por garantir a integração do controle. Este componente constitui a parte não variável desta camada. Os subsistemas são apresentados nos sub-itens a seguir.

V.3.1 Desenvolvimento dos Testes

Este subsistema compreende as funções de geração de testes, definição das falhas e instrumentação do programa em teste (c.f. fig. IV.1).

A função de **geração de testes** trata da obtenção dos casos de teste, esta obtenção podendo se dar de maneira manual ou automática. Na geração manual, os casos de testes são fornecidos pelo usuário. Na geração automática, os casos de teste são obtidos a partir da especificação do sistema. Um aspecto importante na geração dos testes diz respeito à representação dos mesmos. Por exemplo, no caso de testes de protocolos [Vuong93], utiliza-se a notação TTCN (Tree and Tabular Combined Notation) para representar os casos de teste; os dados trocados pelas entidades de protocolo são representados na notação ASN.1 (Abstract Syntax 1), ambas as representações tendo sido padronizadas pela ISO.

A função de **definição das falhas** tem por objetivo auxiliar o usuário na definição das classes de falhas a serem consideradas para injeção, seus atributos e o método para injetá-las.

A função de **instrumentação da implementação** tem como objetivo auxiliar o usuário na preparação do programa para os testes. Dependendo da forma de injeção escolhida (c.f. II.2.5), a função de injeção pode estar embutida no código, o que requer sua inclusão no programa.

V.3.2 Geração de Script

Este subsistema permite que o usuário descreva como a seqüência de testes deve ser realizada. Aseqüência de testes é organizada em **grupos**, cada qual constituído de vários **experimentos** (c.f. II.1.2). O script contém as seguintes informações:

- os casos de teste a serem aplicados e os dados que eles utilizam,
- as classes de falhas a serem injetadas,

- o número de falhas a serem injetadas,
- o momento de injeção,
- o número de nós do sistema, e quais nós serão sumetidos à injeção,
- os processos a serem executados em cada nó, o que inclui a implementação em teste e os testadores,
- a duração do experimento,
- os resultados a serem coletados.

A estratégia de testes adotada, a qual foi apresentada em III.3, prevê a realização de testes estatísticos, com o objetivo de obter medidas da eficiência dos mecanismos de tolerância a falhas do sistema. Para garantir a qualidade destas medidas, é preciso realizar um grande número de experimentos. Portanto, é necessário automatizar o encadeamento de múltiplos experimentos. Uma das dificuldades neste encadeamento diz respeito à sincronização e re-inicialização de vários processos, residentes nos diferentes nós. A re-inicialização do sistema pode consistir simplesmente em recarregar os processos, ou pode ser mais complexa, exigindo o "reset"[8] de todos os nós. A definição de como encadear os experimentos também é parte da geração do script.

V.3.3 Controle de Execução

Este subsistema tem como funções:

- controlar a execução dos testes: esta função é responsável pela ativação e interação com o sistema em teste (*sistema alvo*), de acordo com o script gerado;
- monitorar o sistema alvo: esta função consiste em coletar os dados observados durante os testes nos diversos nós do sistema alvo, e armazená-los para análise a posteriori.

Os dados coletados durante a execução dos experimentos incluem:

- as *interações* (entradas e saídas) observadas,
- o *histórico* dos testes realizados, contendo, entre outras, a identificação dos testes realizados, dos métodos utilizados para obtê-los, os resultados obtidos, a versão da implementação submetida aos testes,
- as *informações de auxílio ao diagnóstico* dos erros existentes na implementação e que venham a ser detectados durante os testes: "dumps" de memória, valores de variáveis de contexto, etc.

Além disso, esse subsistema deve permitir que o usuário possa interagir com o sistema durante a execução dos experimentos, fornecendo-lhe os comandos que permitam obter informações parciais sobre os dados coletados, bem como interromper, recomeçar ou abortar os experimentos.

V.3.4 Tratamento dos resultados

---

[8] Por "reset" entenda-se a re-inicialização total do nó, incluindo a carga do sistema operacional.

Este componente é responsável pelo tratamento dos dados coletados durante os testes, e compreende as seguintes funções: análise do comportamento observado e obtenção de estatísticas.

A **análise do comportamento** tem por objetivo verificar se o comportamento observado está de acordo com o que foi especificado. Esta função pode ser feita manualmente, ou seja, o usuário verifica se os resultados obtidos são os resultados esperados. No entanto, quando o número de experimentos se torna elevado, é necessário que a análise possa ser feita automaticamente.

A **obtenção de estatísticas** vai permitir ao usuário definir os parâmetros de qualidade que deseja obter (fator de cobertura, latência, por exemplo), bem como o método de estimação a ser empregado para obtê-los. Aplicando-se o método escolhido o usuário terá não somente o valor desejado, mas também o erro associado à estimativa.

Além disso, é possível ao usuário visualizar, através de gráficos, os principais resultados das análises realizadas.

V.4 Gerenciamento de Objetos

Esta camada controla os itens de informação produzidos ou manipulados pelos subsistemas mencionados na seção anterior. Esses itens são organizados como *objetos de configuração* [Pressman 92, cap. 21 e 23]. Estes objetos, em geral, não existem isoladamente: eles são compostos por outros objetos, ou são criados a partir de outros, por exemplo. Portanto, conhecer o relacionamento entre estes objetos é também importante.

A camada de Gerenciamento de Objetos provê serviços que permitam aos subsistemas a definição, manipulação e manutenção dos objetos de configuração do ATIFS e dos relacionamentos entre eles. Para garantir a integração de dados mencionada no início da seção, é necessário que esta camada:

- permita que os diferentes subsistemas possam trocar dados entre si, sem maiores dificuldades,
- garanta a consistência dos dados, de modo que as ações realizadas por um susbsistema sobre um determinado objeto não violem as restrições semânticas impostas sobre esse objeto por outro susbsistema,
- torne transparente aos subsistemas do ATIFS, qual o sistema de gerenciamento de banco de dados utilizado;
- faça o controle de versões dos objetos de configuração.

V.5 Gerenciamento e armazenamento dos dados

Esta camada permite ao Gerenciamento de Objetos interagir com o banco de dados, no qual são armazenadas não somente as informações que dão suporte ao desenvolvimento dos testes (especificação do sistema, casos de teste gerado, script, etc), bem como os dados coletados durante os testes, mencionados em V.3.3.

O armazenamento destes dados tem por objetivo permitir ao usuário: (i) comparar a eficiência dos diferentes métodos de teste, em termos do número de erros encontrados na implementação quando da aplicação dos mesmos; (ii) acompanhar a evolução das diferentes

versões da implementação ao longo dos testes; desta forma é possível, por exemplo, determinar se não houve regressão, i.e., se porventura as correções realizadas não introduziram novos erros.

VI. CONCLUSÃO

Este estudo apresentou uma estratégia de teste combinando injeção de falhas e teste formal. Por teste formal entende-se aqui um teste funcional, onde as entradas de teste são derivadas a partir de um modelo do sistema. A integração desses métodos vai permitir complementar as potencialidades de cada uma das técnicas da seguinte forma:

- testes formais ⇨ injeção de falhas: o uso de um modelo formal "guia" o processo de injeção de falhas, servindo:
  - (i) para a derivação das entradas de teste,
  - (ii) como referência para se determinar a correção dos resultados observados,
  - (iii) para determinar a cobertura dos testes.
- injeção de falhas ⇨ testes formais: vai permitir testar um sistema em presença de situações de erro mais amplas, fornecendo também os meios para obtenção de parâmetros necessários na determinação da confiabilidade.

Uma outra característica importante da estratégia proposta é que ela vai permitir combinar duas formas de geração de testes: determinística e estatística, atendendo assim aos dois aspectos da validação, quais sejam: a verificação e a avaliação de parâmetros da eficiência de mecanismos de tolerância a falhas do sistema, o que não ocorre na maioria dos ambientes de testes por injeção de falhas até então.

O texto apresentou também os requisitos para a construção do ATIFS, um ambiente que dê suporte à estratégia proposta. As principais características deste ambiente são:

- suporte às atividades do processo de testes, quais sejam: desenvolvimento e execução dos testes, seguido da análise dos resultados, compreendendo tanto a verificação do comportamento do sistema quanto a avaliação da eficiência de seus mecanismos de tolerância a falhas;
- integração dos diversos subsistemas que realizam estas atividades, integração esta relativa à apresentação (o usuário interage com os diferentes subsistemas de maneira similar), ao controle (os subsistemas utilizam serviços uns dos outros) e aos dados (dados definidos por um subsistema podem ser usados por outros sem maiores dificuldades);
- extensibilidade, i.e., facilidade de se modificar ou acrescentar subsistemas ao ambiente.

Uma versão preliminar desse ambiente será utilizada na validação de uma implementação do protocolo X.25, realizada pelo INPE no âmbito do projeto SCOM-X25, e será implementada em microcomputador do tipo IBM PC.

Para o desenvolvimento desta versão preliminar, os seguintes pontos serão estudados:

- geração das entradas de teste: como determinar os testes a serem aplicados, com base em um modelo na forma de máquina de estados finita? como selecionar as falhas que serão aplicadas durante os testes?
- execução dos testes: como controlar e observar o sistema durante os testes? que arquitetura de testes deve ser utilizada, que seja facilmente adaptável aos diferentes tipos de testes a serem suportados pelo ambiente? como introduzir as falhas no sistema, dado que, nesta versão preliminar, somente as falhas de comunicação serão consideradas?
- análise do comportamento: como obter um oráculo, a partir da especificação do sistema, que leve em conta a presença de falhas, as quais são introduzidas de maneira assíncrona durante a aplicação das entradas de testes?
- organização dos testes: como estruturar os testes a serem aplicados, com base em técnicas da teoria de amostragem, de forma a reduzir os custos (em termos de tempo e recursos) necessários para se obter resultados significativos?

Na próxima versão do ATIFS será levado em conta o uso de uma linguagem para especificar os testes, que seja específica para este ambiente, e que leve em conta os dados e as falhas a serem introduzidos durante os testes.

Uma extensão do ambiente que já está sendo cogitada consiste em ampliar os tipos de falhas consideradas, ou seja, além das falhas de comunicação, ter-se-á as falhas de processador, de memória e até falhas de software, que vão permitir testar mecanismos que tolerem este tipo de falhas.

AGRADECIMENTOS

Gostaria de agradecer a Ana Maria Ambrósio pela sua colaboração na revisão deste texto, e também ao Marcio Roberto Stefani pelas "dicas" de português. Estas colaborações contribuíram bastante para a melhoria da qualidade deste trabalho.

REFERÊNCIAS

[Arlat 89] J.Arlat, Y.Crouzet, J.-C.Laprie. Fault injection for dependability validationof fault-tolerant computing systems. *Proc. FTCS-19*, Chicago, IL, USA, jun. 1989.

[Arlat 90a] J.Arlat. *Validation de la sûreté de fonctionnement par injection de fautes: méthode - mise en-œuvre - application*. Tese de doutorado de estado, INPT, 1990.

[Arlat 90b] J.Arlat, M.Aguera, L.Amat, Y.Crouzet, J.-C.Fabre, J.-C.Laprie, E.Martins, D.Powell. Fault injection for dependability validation - a methodology and some applications. *IEEE Transactions on Software Engineering*, vol. 16, fev. 1990.

[Arlat 91] J.Arlat, Y.Crouzet, E.Martins, D.Powell. Dependability testing report LA3 - fault injection on the Extended Self-Checking NAC. *Relatório interno do projeto Delta-4,* n° 91.396, dez. 1991.

[Arnold 73] Thomas F.Arnold. The concept of coverage and its effect on the reliability model of a repairable system. *IEEE Transactions on Computers*, C-22(3), 1973, pp. 251-254.

[Avresky 91] D.R.Avresky, J.Arlat, J.-C.Laprie, Y.Crouzet. Guiding the process of fault injection for testing fault tolerance. *Relatório interno LAAS n° 91-351*. Dezembro 1991.

[Ayache 79] J.M.Ayache, P.Azema, M.Diaz. Observer: a concept for on-line detection of control errors in concurrent systems. *Proc. FTCS-9*, Wisconsin, Madison, USA, 1979.

[Bochmann 89] G.v.Bochmann, R.Dssouli, J.R.Zhao. Trace analysis for conformance and arbitration testing. *IEEE Transactions on Software Engineering*, 15(11), 1989.

[Bouricius 69] W.G.Bouricius, W.C.Carter, P.R.Schneider. Reliability modeling techniques for self-repairing computer systems. *Proc. 24th. National Conference of ACM*, 1969.

[Crouzet 82] Y.Crouzet, B.Decouty. Measurements of fault detection mechanisms efficiency: results. *Proc. FTCS-12*, Santa Monica, CA, USA, jun 1982.

[Czeck 90] E.W.Czeck, D.P.Siewiorek. Effects of transient gate-level faults on program behavior. *Proc. FTCS-20*, Newcastle upon Tyne, Inglaterra, jun 1990.

[Damm 88] A.Damm. *Experimental evaluation of error-detection and self-checking coverage of components of a distributed real-time system*. Tese de doutorado, Tech. Univ. Viena, 1988.

[Dilenno 91] T.R.Dilenno, D.A.Yaskin, J.H.Barton. Fault tolerance testing in the Advanced Automation System. *Proc. FTCS-21*, Montreal, Canadá, jun 1991.

[Echtle 92] K.Echtle, M.Leu. The EFA fault injector for fault-tolerant distributed system testing. Anais do IEEE *Workshop on Fault Tolerant Parallel and Distributed Systems*, Amherst, MA, EUA, 1992.

[Goswami 91] K.K.Goswami, R.K.Iyer, "DEPEND: a simulation based environment for system level dependability analysis". *Relatório da Univ. of Illinois at Urbana-Champaign* n° CRHC-UIUC, 1991.

[Green 72] A.E.Green, A.J.Bourne. *Reliability Technology,* John Wiley & Sons, Inglaterra, 1972.

[Gunneflo 89] U.Gunneflo, J.Karlsson, J.Torin. Evaluation of error detection schemes using fault injection by heavy-ion radiation. *Proc. FTCS-19*, Chicago, IL, USA, 1989.

[Heimerdinger 92] W.L.Heimerdinger. A conceptual framework for system fault tolerance. *Relatório Interno* CMU/SEI-92-TR-33, Software Engineering Institute, Universidade Carnegie-Mellon. out/92.

[Jenn 92] E.Jenn, J.Arlat, "Implementation of fault injection in VHDL". *Relatório Interno* n° LAAS92266, julho de 1992. (em francês)

[Kanawati 92] G.A.Kanawati, N.A.Kanawati, J.A.Abraham. FERRARI: A Tool for the Validation of System Dependability Properties. *Proc. FTCS-22*, Boston, MA, USA, 1992.

[Kao 93] W.Kao, R.K.Iyer, D.Tang. FINE: A fault injection and monitoring environment for tracing the Unix system behavior under faults. *IEEE Transactions on Software Engineering*, 19(11), 1993.

[Katsuyama 91] K.Katsuyama, F.Sato, T.Nakakawaji, T.Mizuno. Strategic testing environment with formal description techniques. *IEEE Transactions on Computers*, 40(4), 1991.

[Kopka 88] B.Kopka. Etude et validation d'une redondance homogène d'ordre deux à décalage temporel pour des applications à haut niveau de sécurité. Tese de doutorado, Nancy, França, 1988.

[Kurlak 81] R.P.Kurlak, J.Chobot. CPU coverage evaluation using automatic fault injection. *American Institute of Aeronautics and Astronautics*, 1981.

[Lala 83] J.H.Lala. Fault detection, isolation and reconfiguration in FTMP: methods and experimental results. *5th. IEEE/AIAA Digital Avionics System Conference*, 1983.

[Laprie 92] J.-C.Laprie. Sûreté de fonctionnement: concepts de base et terminologie. *Dependable Computing and Fault Tolerance*. Springer Verlag, 1992.

[Legrand 90] L.Legrand, G.Sandres. Le test des protocoles de télécommunication et les outils XRTLE. *Génie Logiciel & Systèmes Experts*, 18(3), 1990.

[Leite 87] Julius C.B.Leite, Orlando G.Loques F°. Software. *II SCTF*, cap. 4 do mini-curso intitulado: Introdução à Tolerância a Falhas, Campinas, SP, 1987.

[Lovric 93] T.Lovric, K.Echtle. ProFI: Processor Fault Injection for dependability validation. *IEEE Intl. Workshop on Fault and Error Injection for Dependability Validation of Computer Systems*, Gotemburgo, Suécia, 1993.

[Madeira 93] H.Madeira, F.Moreira, M.Rela, P.Furtado, J.G.Silva. Pin-level fault injection for dependability validation: some research results at the University of Coimbra. *IEEE Intntl. Workshop on Fault and Error Injection for Dependability Validation of Computer Systems*, Gotemburgo, Suécia, 1993.

[Martins 89] E.Martins, J.Arlat, Y.Crouzet, J.C.Fabre, D.Powell. Testing multipeer protocols in the presence of faults. *Proc. IFIP TC6 2nd. Workshop on Protocol Test Systems*, Berlim, Alemanha, 1989.

[Martins 93] E.Martins. Injeção de falhas na validação experimental da tolerância a falhas. *V SCTF*, mini-curso, S.José dos Campos, Outubro de 1993.

[Pressman 92] R.S.Pressman. Software Engineering: a Practitioners Approach. McGraw-Hill International Editions, 1992, 3a. ed.

[Rosenberg 93] H.A.Rosenberg, K.G.Shin. Software Fault Injection and its Application in Distributed Systems. *Proc. FTCS-23*, Toulouse, França, 1993.

[Segall 88a] Z.Segall, D.Vrsalovic, D.P.Siewiorek, D.Yaskin, J.Kownacki, J.Barton, R.Dancey, A.Robinson, T.Lin. FIAT-Fault Injection based Automated Testing environment. *Proc. FTCS-18*, Tokyo, Japão, jun 1988.

[Segall 88b] Z.Segall, J.Barton, D.Vrsalovic, D.P.Siewiorek, R.Dancey, A.Robinson. Fault injection based automated Testing: practice and examples. *Proc.8° Digital Avionics System Conference*, San Jose, EUA, 1988.

[Sidhu 89] D.P.Sidhu, C.S.Chang. Probabilistic testing of protocols. *ACM SIGCOMM'89*, Austin, Texas, USA, set. 1989.

[Thévenod 91a] P.Thévenod-Fosse. From random testing of hardware to statistical testing of software. *IEEE Comp'Euro91*, Bolonha, Itália, 1991.

[Thévenod 91b] P.Thévenod-Fosse, H.Waeselynck. An investigation of software statistical testing. *Relatório Interno* LAAS n° 91.003, 1991.

[Thévenod 92] P.Thévenod-Fosse, H.Waeselynck. On functional statistical testing designed from software behavior models. *3° IFIP Intl. Working Conference on Dependable Computing for Critica1 Applications (DCCA-3)*, Palermo, Itália, 1992.

[Thomas 92] I.Thomas, B.A. Nejmeh. SDefinition of tool integration for environments. *IEEE Software*, mar/1992.

[Vuong 93] S.T.Vuong. Protocol conformance testing. Tutoriais e Mini-cursos do *SBRC-11,* Campinas, SP, 1993.

[Vuong 94] S.T.Vuong, H.Janssen, Y.Lu, C.Mathieson, B.Do. TESTGEN: an environment for protocol test suite generation and selection. *Computer Communications*, 17 (4), 1994.

[Weyuker 82] E.J.Weyuker. On testing non-testable programs. *The Computer Journals*, 10(2), 1987.

[Winfrey 89] T.L.Winfrey, G.E.Kaiser. Testing reliable distributed applications through simulated events. *Proc. FTCS-19*,1989.

[Zeng 86] H.X.Zeng, D.Rayner. The impact of the ferry concept on protocol testing. *Protocol Specification, Testing and Verification*, editado por M.Diaz. North-Holland, 1986.