Utilização de Reflexão Computacional em Tempo de Compilação para Implementação de Ferramentas de Injeção de Falhas

Michael Menna Barreto Leske
mileske@dcc.unicamp.br

Eliane Martins
eliane@dcc.unicamp.br

Resumo

Existem atualmente diversos tipos de ferramentas para injeção de falhas por software. Muitas delas, no entanto, apresentam um destes problemas: ou causam um overhead muito grande na aplicação alvo ou só conseguem representar um modelo de falhas significativo em um nível muito baixo, muito próximo do hardware mas muito distante da aplicação. Neste artigo, apresenta-se um técnica de implementação de ferramentas de injeção de falhas que procura contrabalançar estes dois fatores.

Abstract

There are many kinds of software fault injection tools nowadays. Many of them, however, present one of the following problems: either they cause a great overhead on the target application, or they can only represent a significant fault model in a very low level, near the hardware, but too distant from the application. This article presents a technique for implementing fault injection tools that counterbalances these two factors.

Universidade Estadual de Campinas - SP

Instituto de Computação

I. Introdução

Neste artigo propõe-se o uso de uma ferramenta de injeção de falhas (FIRE) que utiliza reflexão computacional com o objetivo de reduzir o custo associado à injeção. A arquitetura reflexiva utilizada é estrutural, onde a estrutura estática das classes da aplicação alvo é alterada, de forma a se implementar o comportamento desejado.

O artigo está dividido da seguinte forma: a seção 2 apresenta a terminologia utilizada em injeção de falhas; a seção 3 apresenta a técnica de injeção de falhas como forma de validar sistemas tolerantes a falhas; a seção 4, o conceito de reflexão computacional bem como a linguagem reflexiva utilizada na implementação da ferramenta em questão; na seção 5 é apresentada a ferramenta FIRE; finalmente, a seção 6 apresenta uma breve conclusão.

II. Terminologia

A terminologia em português na área de tolerância a falhas é ainda bastante controversa. Os termos aqui apresentados estão baseados em [1] e [4].

- segurança de funcionamento (*dependability*): pode ser definida como a qualidade do serviço fornecido pelo sistema, tal que a confiança possa ser justificadamante nele depositada. De acordo com [2], as duas principais medidas da segurança de funcionamento são:
  - confiabilidade (*reliability*): probabilidade do sistema não apresentar defeitos em um intervalo de tempo;
  - disponibilidade (*availability*): probabilidade do sistema estar operando em um determinado intervalo de tempo;

- falha (*fault*): é a causa direta de um erro;

- erro (*error*): parte do estado errôneo do sistema que constitui a diferença em relação ao estado válido [1];

- defeito (*failure*):um defeito em um sistema ocorre quando seu comportamento desvia daquele requerido pela sua especificação;

- validação: o processo de validação da segurança de funcionamento consiste na verificação, para retirada de falhas, e na avaliação, para estimar medidas de confiabilidade, disponibilidade, eficiência dos Mecanismos de Tolerância a Falhas (MTF's), entre outras.

III - Injeção de Falhas

A segurança de funcionamento de um sistema computacional tolerante a falhas deve ser validada de forma a garantir que sua redundância foi corretamente implementada e o sistema fornecerá o nível desejado de confiabilidade. A injeção de falhas - a inserção deliberada de falhas em um sistema para determinar sua resposta - representa uma solução eficiente para este problema. A seguir são apresentados os principais tipos de falhas.

Falhas de hardware: são classificadas principalmente quanto à duração [2]

- falhas permanentes: causadas por danos irreversíveis em um certo componente. Uma vez que tenha ocorrido uma falha permanente, o componente com a falha deve ser substituído ou, se possível, reparado;
- falhas transientes: são causadas por distúrbios ambientais, como flutuações de voltagem, interferência eletromagnética ou radiação. Geralmente são de curta duração e não causam danos permanentes (ainda que possam levar o sistema a um estado errôneo);
- falhas intermitentes: tendem a oscilar entre períodos de atividade errônea e de dormência. Geralmente são causadas por erros de projeto, que resultam em hardware instável.

Falhas de hardware que conduzem a erros de software: são erros de software induzidos pelo hardware. Podem ser classificadas em: falhas de memória, falhas de processador, falhas de barramentos, falhas de I/O e falhas de comunicação;

Falhas de Software: são causadas por problemas na especificação, no projeto ou na codificação do programa. Podem ser classificadas em: falhas de inicialização, falhas de atribuição e falhas de decisão.

Uma vez caracterizada a injeção de falhas e apresentados os principais tipos de falhas, será apresentada a técnica de injeção de falhas por software.

III.1 - Injeção de falhas por software

Na injeção de falhas por software (ou simplesmente, injeção por software), falhas lógicas são introduzidas em um protótipo de software do sistema, com o objetivo de emular a conseqüência de falhas físicas. Esta técnica apresenta as seguintes vantagens em relação à injeção física:

- facilidade de realização, pois não necessita de equipamento especial de hardware;
- a controlabilidade e observabilidade do sistema durante os testes é mais fácil.

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

Entretanto esta é uma técnica extremamente intrusiva, afetando o tempo de resposta da aplicação alvo. Na seção VII será apresentada a ferramenta FIRE, que através da reflexão computacional minimiza este efeito.

A injeção por software pode ser classificada em dois tipos: injeção em tempo de compilação e injeção em tempo de execução. Detalhes sobre estes tipos de injeção podem ser encontrados em [2] e [3].

IV - Reflexão Computacional

Reflexão computacional é a atividade executada por um sistema quando este faz computações sobre suas próprias computações [7]. O sistema é dividido em dois níveis: nível base, no qual implementa-se a funcionalidade da aplicação, e meta-nível, que observa e manipula a estrutura e/ou comportamento do nível base.

Entre as vantagens da utilização dessa técnica na injeção de falhas por software, pode-se citar:

- redução de perturbação no código da aplicação alvo através da separação entre o nível base e o meta-nível (permitindo independência entre a funcionalidade da aplicação e a injeção/monitorização);
- a implementação da injeção/monitorização em meta-nível permite a reutilização de componentes, pois ela independe da aplicação;
- flexibilidade na injeção de falhas devido ao uso de metaobjetos, que podem injetar diferentes tipos de falhas em diversos objetos do nível-base.

A linguagem reflexiva escolhida para implementação da ferramenta FIRE foi o OpenC++ 2,5.

IV.1 - OpenC++ 2.5

O OpenC++ 2.5 é uma ferramenta para o desenvolvimento de tradutores e analisadores de código C++. O programador que deseja utilizar o OpenC++ 2.5 deve escrever um programa em meta-nível que especifica como traduzir ou analisar um programa C++. A seguir o programa em meta-nível é compilado pelo compilador OpenC++ e ligado (estática ou dinamicamente) ao compilador como um *plug-in*. O compilador resultante traduz ou analisa o programa fonte (que constitui o nível-base) de acordo com o especificado pelo meta-nível.

O meta-nível é escrito de acordo com uma interface, que constitui o protocolo de Metaobjeto do OpenC++ 2.5 (PMO).

O nível base é primeiramente pré-processado pelo pré-processador C++ e depois dividido em pequenos pedaços de código. Estes pedaços são traduzidos por metaobjetos e remontados, formando um programa C++ completo. No OpenC++ 2.5, estes pedaços de código são representados por metaobjetos *Ptree* (parse tree). Ainda que os metaobjetos sejam idênticos a objetos C++ convencionais, eles fazem parte do compilador e representam meta-características do nível base (daí serem denominados metaobjetos).

A arquitetura da OpenC++ 2.5 é certamente diferente da maioria das arquiteturas reflexivas. Entretanto, o metaobjeto continua controlando o comportamento do nível-base. A diferença é que o nível base não é interpretado da maneira convencional, mas sim traduzido em tempo de compilação de forma que o comportamento desejado seja implementado. Ou seja, o meta-nível diz como o nível base deve ser traduzido e o compilador OpenC++ realiza esta tradução. Ao final do processo, tem-se um programa C++ convencional, onde o comportamento desejado foi implementado (nas arquiteturas reflexivas convencionais, o meta-nível altera o comportamento do nível-base em tempo de execução). Como em outras linguagens reflexivas, a classe do metaobjeto tem um membro para cada ação básica do objeto, como por exemplo chamadas de métodos, escrita e leitura de dados, criação de objetos e assim por diante.

V. A ferramenta Fire

A FIRE (Fault Injection Using a Reflexive Architecture) [4] é uma ferramenta de injeção de falhas por software que utiliza reflexão computacional (seção 4) para injetar falhas e monitorizar seus efeitos. Originalmente, ela foi desenvolvida utilizando-se uma versão do OpenC++ que possuía uma arquitetura reflexiva convencional. Aqui é apresentada uma nova versão da ferramenta, implementada com a versão 2.5 do OpenC++.

Entre as principais qualidades da FIRE, pode-se citar: modularidade, a fim de facilitar a incorporação de novas características; reusabilidade, para permitir fácil adaptação em aplicações alvo

diferentes, e portabilidade, para permitir sua utilização em plataformas (hardware/software) diferentes com alterações mínimas.

A arquitetura da ferramenta FIRE é apresentada na figura 1. Seus principais componentes são: uma aplicação alvo, uma biblioteca de meta-nível, um controlador e uma interface com o usuário.

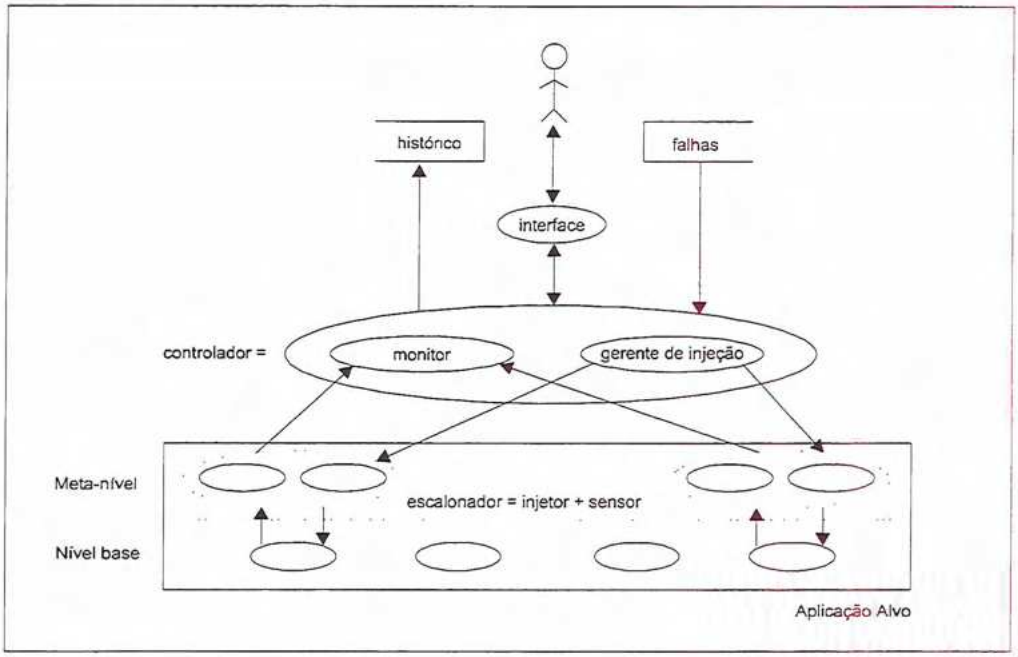

Figura 1: Arquitetura da ferramenta FIRE

A aplicação alvo é uma aplicação orientada a objetos, cujos componentes tolerantes a falhas devem ser validados. Ela deve ser submetida a uma fase de instrumentação, na qual as diretivas de reflexão são introduzidas.

A biblioteca de meta-nível é anexada à aplicação alvo em tempo de compilação e define meta-objetos denominados escalonadores, compostos por um injetor e um sensor. O primeiro corrompe valores do objeto base e o segundo coleta dados sobre o processo de injeção de falhas.

O controlador coordena o experimento: inicia o processo da aplicação, controla injetores e sensores e armazena dados coletados. É composto por um gerente de injeção, que lê a falha a ser injetada do arquivo de falhas e a transfere ao escalonador (que por sua vez a transfere ao injetor), e por um monitor, que recebe os dados recebidos pelo sensor e os armazena no arquivo de histórico do experimento.

A interface do usuário auxilia na observação e manutenção dos experimentos.

V.1 - Aspectos de Implementação

É necessário de alguma forma estabelecer uma conexão entre a aplicação a ser testada e a biblioteca de meta-nível, que contém os métodos que vão realizar a injeção de falhas em tempo de execução. Para fazer esta conexão utiliza-se um módulo da ferramenta FIRE implementado utilizando-se a linguagem OpenC++ 2.5.

A partir de uma aplicação previamente instrumentada, este módulo insere novo atributo em cada classe que sofrerá injeção. Este atributo chama-se *metaobj* e é da classe *SchedulerMetaObj*, que implementa a biblioteca de meta-nível.

Há três tipos de injeção permitidos pela ferramenta: injeção em argumentos de métodos, injeção em valores de retorno e injeção em argumentos. Para cada tipo, o módulo de conexão tem um comportamento distinto.

Injeção em Argumentos de Métodos

Para cada método a sofrer injeção, é inserida uma chamada a metaobj.Inject(), em que se passa por referência o argumento a sofrer injeção. Assim, toda vez que o método em questão é chamado, o argumento desejado é corrompido.

Injeção em Valores de Retorno

Para fazer injeção em valores de retorno, o módulo de conexão faz um parse do método em questão e substitui toda a ocorrência de "**return** *expr*" por "**return** metaobj.Inject_re(*expr*)". O método *metaobj.Inject_re()* retorna a expressão corrompida.

Injeção em Atributos

Para realizar injeção em atributos, o módulo de conexão substiui toda a ocorrência do atributo em questão por "metaobj.Inject_at(*atributo*)". O método *metaobj.Inject_at()* retorna o valor do atributo corrompido.

No quadro abaixo é apresentado um pequeno trecho de uma aplicação instrumentada, antes e depois de se ter efetuado a conexão com a biblioteca de meta-nível:

Antes da Conexão	Após a Conexão

metaclass MetaClass Person:
class Person {
public:
Person(int);
inject_re int BirthdayComes();
private:
int age;
};
 	
Person::Person(int i)
{
age=i;
}
 	
int Person::BirthdayComes()
{
age++;
return age;
}
	class Person {
	public:
	Person(int);
	int BirthdayComes();
	private:
	int age;
	SchedulerMetaObj metaobj;
	};
	
	Person::Person(int i)
	{
	age=i;
	}
	
	int Person::BirthdayComes()
	{
	age++;
	return metaobj.Inject_re (age);
	}

A instrumentação da aplicação-alvo é bastante simples e consiste basicamente em especificar quais classes estarão sujeitas à injeção e onde ela ocorrerá (neste exemplo, ela ocorre no valor de retorno do método BirthdayComes). As linhas em negrito correspondem às alterações efetuadas na aplicação alvo pelo módulo de conexão com o meta-nível. A injeção de falhas ocorrerá através do método Inject_re, cujo comportamento dependerá do arquivo de falhas fornecido ao gerente de injeção, que define, entre outras coisas, o padrão de repetição da falha (permanente, intermitente ou transiente) e a operação a ser realizada para corromper o alvo.

VI - Conclusão

A ferramenta FIRE consegue representar falhas em troca de mensagens e falhas em atributos de classes. Estas últimas podem simular, por exemplo, uma falha de hardware que causa uma leitura errônea do atributo. Desta forma, a ferramenta consegue representar falhas de alto nível.

O custo da injeção de falhas utilizando a FIRE consiste no custo associado a chamada dos métodos responsáveis pela injeção. Ainda que em algumas circunstâncias este custo possa representar um *overhead* inaceitável, ele não é significativo para a maioria das aplicações.

Bibliografia:

[1] Beder, D. M, "Integração dos Mecanismos de Recuperação de Erros por Avanço e por Retrocesso". Dissertação de Mestrado. IC - UNICAMP - Campinas.

[2] Clark, J. A. & Pradhan, D. K, "Fault Injection: A Method for Validating Computer System Dependability". *IEEE Computer*, jun. 1995, pp. 47-56.

[3] Hsueh, M, Tsai, T. K & Iyer, R. K, "Fault Injection Techniques and Tools". *IEEE Computer*, abr. 1997, pp 52-75.

[4] Rosa, A. C. A, "Uma Arquitetura Reflexiva para Injetar Falhas em Aplicações Orientadas a Objetos". Dissertação de Mestrado. IC - UNICAMP - Campinas.

[5] Rosa, A. C. A. & Martins, Eliane, "Using Reflective Programming to Inject Faults into Object Oriented Systems". *Proc. of the 1998 IFIP International Workshop on Dependable Computing and its Applications*, Johannesburg, South Africa, jan. 12 - 14, 1998, pp. 227 - 236.

[6] Chiba, S, "OpenC++ 2.5 Reference Manual". Institue of Information Science and Eletronics. University of Tsuukuba. 1997-99

[7] Lisbôa,M. L. B, "Reflexão Computacional no Modelo de Objetos". Universidade Federal do Rio Grande do Sul, ago. 1997