Utilizando metaobjetos para injetar falhas e monitorizar seus efeitos

Amanda Cibele Apolinário Rosa*, Eliane Martins
Instituto de Computação - Universidade Estadual de Campinas (Unicamp)
P.O Box 6176 Campinas - 13083-970 SP Brasil
{amandapo, eliane}@dcc.unicamp.br

Abstract

Software-implemented fault injection is, nowadays, a largely used technique to validate dependability properties of software systems. To inject faults and monitor their effects some form of instrumentation may be introduced into the system under test (target system). This instrumentation causes some level of intrusiveness, i.e., it imposes some interference upon the target system execution. Therefore, a goal of a software instrumentation approach is to be the least intrusive possible. To obtain this quality we propose in this study the use of reflective object-oriented programming. Intrusiveness in the target system is reduced, in that it allows a clear separation between functional and non-functional aspects, the later being related to fault injection and monitoring aspects. This papers describes a reflective test architecture, shows how faults are injected using reflection and presents some results of experiments.

Resumo

Injeção de falhas por software é uma técnica que vem sendo muito utilizada para validar as propriedades de segurança no funcionamento de sistemas de software. Para injetar falhas e monitorizar seus efeitos alguma forma de instrumentação deve ser introduzida na aplicação em teste (aplicação alvo). Essa instrumentação é intrusiva, ou seja, interfere na execução da aplicação alvo. No entanto, um dos objetivos de uma abordagem de instrumentação de software é ser o menos intrusiva possível. Para alcançar este objetivo neste estudo nós propomos o uso da programação orientada a objetos reflexiva. Reflexão reduz a interferência na aplicação alvo porque provê uma separação clara entre os aspectos funcionais e não-funcionais, sendo o último relacionado aos aspectos de injeção de falhas e monitorização. Este artigo descreve uma arquitetura reflexiva de teste, mostra como injetar falhas utilizando reflexão e apresenta alguns resultados de experimentos.

* Trabalho parcialmente financiado pela FAPESP.

1. Introdução

Injeção de falhas por software é uma técnica que vem sendo muito utilizada para validar as propriedades de segurança no funcionamento de sistemas de software e diversas ferramentas para tal propósito têm sido desenvolvidas. Tais ferramentas utilizam abordagens diferentes para realizar a injeção de falhas, algumas dessas abordagens são: inserção de instruções de *trap* (exceções de software) no código fonte alvo, execução da aplicação alvo no modo traço ou no modo privilegiado e utilização de características especiais do hardware.Uma boa apresentação da área é feita em [HTI97]. Apesar do grande número de ferramentas, a injeção de falhas por software tem suas limitações. Uma delas é a perturbação na aplicação em teste: a fim de injetar falhas e monitorizar seus efeitos alguma forma de instrumentação é introduzida no código fonte da aplicação alvo. Essa instrumentação interfere na execução da aplicação e/ou altera a estrutura original do seu código fonte. Em nosso trabalho nós propomos o uso de reflexão computacional para minimizar a perturbação no software da aplicação alvo nos testes de injeção de falhas por software em aplicações orientadas a objetos. Reflexão foi escolhida por ser uma maneira simples de separar a funcionalidade da instrumentação da funcionalidade da aplicação alvo. Reflexão introduz um modelo de arquitetura no qual existe o nível base e o meta-nível. O nível base neste estudo é usado para implementar objetos da aplicação alvo. O meta-nível permite a programadores observar e manipular dados e ações executadas no nível base e neste estudo é utilizado para implementar as características de injeção de falhas e monitorização.

O restante do texto está organizado da seguinte forma: a seção 2 apresenta rapidamente uma visão sobre programação orientada a objetos reflexiva e suas vantagens para injeção de falhas por software, enfocando a linguagem usada neste estudo, OpenC++ 1.2. A seção 3 descreve nossa arquitetura de teste e apresenta como a abordagem metaobjetos é utilizada para injeção falhas e monitorização. A seção 4 apresenta alguns resultados de experimentos realizados. E finalmente a seção 5 conclui este artigo.

2. Reflexão e orientação a objetos

2.1 Conceito e vantagens

Reflexão Computacional, ou apenas reflexão, é a atividade executada por um sistema computacional quando faz computações sobre suas próprias computações [Mae87]. Reflexão é obtida subdividindo-se o sistema em dois níveis de processamento: o *nível base*, que implementa a funcionalidade da aplicação e o *meta-nível*, que observa e manipula a estrutura e/ou comportamento do nível base.

Patti Maes [Mae87] introduziu a abordagem *metaobjeto* para implementar reflexão em sistemas orientados a objetos. Um objeto x pode ser associado a um metaobjeto ^x que representa a meta-informação de x.

Reflexão e orientação a objetos apresentam diversas vantagens para injeção de falhas por software:

1. Redução da perturbação no código da aplicação alvo. Os requisitos de injeção de falhas e monitorização (encapsulados em meta-objetos) ficam separados dos requisitos associados ao propósito da aplicação (encapsulados em objetos).
2. Reutilização de componentes. A separação de domínios permite que os aspectos de injeção de falhas e monitorização sejam desenvolvidas independentes da aplicação, propiciando a reutilização tanto de objetos da aplicação como de metaobjetos.

3. Flexibilidade na injeção de falhas. O uso de metaobjetos possibilita injetar diferentes tipos de falhas em diferentes objetos do nível base, de acordo com suas características.
4. Facilidade de uso. Metaojetos de injeção de falhas e monitorização podem ser facilmente incorporados e retirados da aplicação em teste.

2.2 A linguagem OpenC++ 1.2

OpenC++ 1.2 [Chi93], uma extensão de C++ que dá apoio à reflexão, é a linguagem que vem sendo utilizada neste trabalho. Suas principais características são [Chi93, Lis97]:

1. Objetos no nível base podem ser reflexivos ou não. Objetos reflexivos podem possuir métodos e atributos reflexivos e são associados a metaobjetos, os quais controlam seu comportamento. Um objeto não-reflexivo é um objeto C++ convencional.
2. Cada objeto reflexivo pode ser associado a um único metaobjeto.
3. Um metaobjeto controla a invocação de métodos reflexivos e o acesso a atributos reflexivos do objeto reflexivo do nível base a ele associado.
4. Um metaobjeto é uma instância de uma classe de meta-nível. Classes de meta-nível são associadas a classes do nível-base em tempo de compilação.
5. Classes do meta-nível são derivadas da classe predefinida *MetaObj*, a qual define os métodos para um metaobjeto controlar o objeto reflexivo a ele associado.
6. Um nome_de_categoria pode ser associado a métodos e atributos reflexivos, permitindo que um metaobjeto altere tais métodos e atributos de acordo com a categoria especificada.
7. Um programa em OpenC++1.2 é um programa C++ que contém diretivas para declarar classes, atributos e métodos como reflexivos. Essas diretivas são incorporadas como comentários C++ que começam com *//MOP*.

Essas características ficarão mais claras na próxima seção com os exemplos mostrando seu uso na arquitetura proposta.

3. Utilizando programação orientada a objetos reflexiva para injeção de falhas e monitorização

Nesta seção fazemos uma descrição rápida da nossa ferramenta; apresentamos a arquitetura reflexiva e descrevemos como é feita a injeção de falhas e a monitorização de seus efeitos.

3.1 Arquitetura

A arquitetura para injeção de falhas e monitorização é apresentada na figura 1. Seus principais componentes são: uma *aplicação alvo* com os mecanismos de tolerância a falhas a serem validados, uma *biblioteca de meta-nível*, um *controlador* e um *módulo de interface com o usuário*.

A *aplicação alvo* é uma aplicação tolerante a falhas orientada a objetos que deve ser submetida a uma fase de instrumentação. Nessa fase são introduzidas as diretivas de reflexão para indicar quais objetos e seus respectivos métodos e atributos serão reflexivos, ou seja, serão injetados e/ou monitorados.

A *biblioteca de meta-nível* deve ser anexada ao sistema alvo em tempo de compilação e define metaobjetos denominados *escalonadores*; um escalonador é composto por um *injetor* e *sensor*[1]. Metaobjetos *escalonadores* capturam a mensagem (chamada de método ou acesso a

[1] O escalonador é então um objeto composto, criado devido a uma limitação da linguagem OpenC++ 1.2: cada objeto pode ser associado a um único metaobjeto. A separação das funções de injeção e monitorização é necessária para fins de modularidade e reusabilidade, justificando assim a necessidade de um objeto composto.

atributo) enviada a um objeto reflexivo, verificam a categoria (monitorização e/ou injeção) associada a mensagem, decidem que tipo de ação deve ser realizada, injeção de falhas, coleta de dados ou computação normal do objeto do nível base e podem delegar operações a um *injetor* ou *sensor*. Um *injetor* corrompe valores do objeto base. Cada *sensor* coleta dados sobre a ativação dos mecanismos de tolerância a falhas e sobre o valor de argumentos e atributos de objetos da aplicação definidos para ser monitorizados, além disso coleta dados sobre o processo de injeção de falhas.

O *controlador* coordena os experimentos: inicia o processo da aplicação, controla (via *escalonador*) *injetores* e *sensores* e armazena os dados coletados. O controlador é composto por dois objetos: um *gerente de injeção* e um *monitor*. O *gerente de injeção* lê a falha ser injetada do arquivo *falhas* e a transfere ao *escalonador* que por sua vez a transfere ao *injetor*. O *monitor* recebe os dados enviados pelos *sensores* e os armazena no arquivo de *histórico* do experimento.

A *interface do usuário* auxilia os testadores a observar e manipular a execução de experimentos.

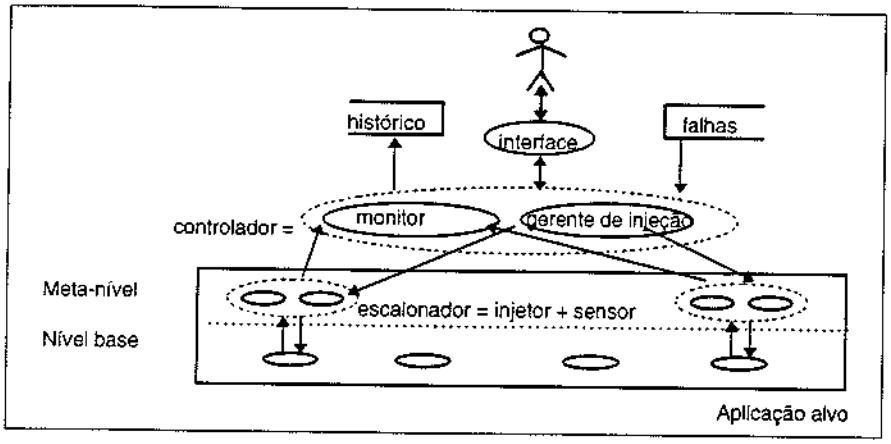

Figura 1: Uma arquitetura reflexiva para injeção de falhas e monitorização.

3.2 Injeção de falhas

Na ferramenta proposta, falhas são injetadas em tempo de execução, assim um mecanismo de ativação das falhas injetadas é necessário. A ativação de falhas pode ser temporal ou espacial [CMS95]. Na ativação temporal, falhas são ativadas quando um tempo predefinido é atingido. Na ativação espacial, falhas podem ser ativadas quando a execução da aplicação passa por um endereço previamente especificado [KKA92], e/ou certos dados são utilizados [CMS95]. Os tipos de falhas considerados são uma outra característica importante de uma ferramenta de injeção de falhas por software. Ambos os aspectos são discutidos a seguir.

3.2.1 Mecanismo de ativação

Em nossa abordagem a ativação das falhas é espacial, utilizando o mecanismo de interceptação de mensagens OpenC++1.2. A intercepação de mensagens é a característica provida pelo protocolo de metaobjeto da linguagem OpenC++1.2 para controlar objetos reflexivos, e pode ocorrer na chamada de métodos reflexivos ou no acesso a atributos reflexivos.

A figura 2 mostra como falhas são ativadas pela chamada de um método reflexivo. Quando um método reflexivo é chamado no nível base, a chamada é capturada e manipulada no meta-nível (por um *escalonador*) através do método *Meta_MethodCall* (①). Este meta-método verifica

se a falha dever ser injetada. Em caso afirmativo invoca um *injetor* para injetar a falha especificada. Então o *escalonador* invoca o método da aplicação usando outro método *Meta_HandleMethodCall* (②③). No final do *Meta_MethodCall*, o controle retorna para o nível base e os resultados são retornados ao chamador na aplicação como em uma chamada de método normal (④). O processo é o mesmo para o acesso a atributos reflexivos.

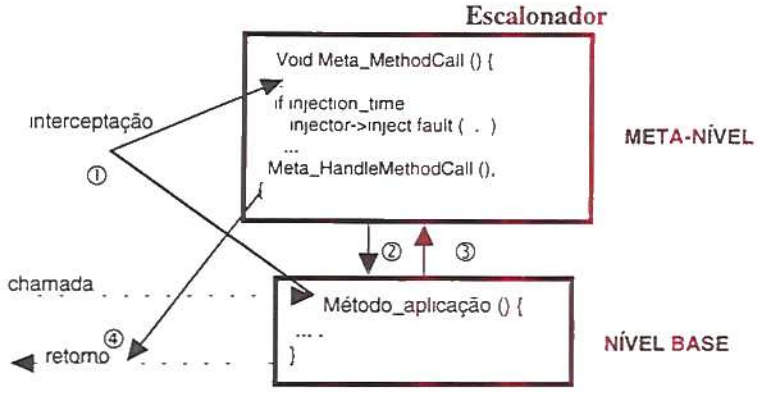

Figura 2: Ativação da injeção via chamada de método reflexivo.

Como um objeto pode ser injetado e/ou monitorado pelo metaobjeto, o *escalonador* precisa de um nome-de-categoria (ver item 2.2) para identificar a ação a ser executada de acordo com o método ou atributo em questão.

3.2.2 Tipos de falhas

Em nossa abordagem de teste, fazemos um paralelo com circuitos de hardware considerando objetos como circuitos de software, como proposto em [Hof89]. Quando consideramos objetos como circuitos, nós podemos usar a mesma correspondência utilizada por Hoffman; ou seja, cada método (ou atributo) público pode corresponder a um pino de circuito, ou a um conjunto relacionado de pinos. Fazendo um paralelo com a injeção de falhas por hardware, podemos injetar falhas internas, afetando as características privadas[2] do objeto, ou falhas externas, afetando a interface do objeto.

Uma outra dimensão da classificação das falhas é baseada no seu padrão de repetição: falhas podem ser transientes (nunca repetem), intermitentes (repetem conforme um intervalo predefinido) ou permanentes (sempre repetem). Nossa ferramenta pode injetar esses três tipos de falhas.

4. Alguns resultados

A fim de validar nossa ferramenta foi realizada uma série de experimentos tendo como alvo a aplicação orientada a objetos tolerante a falhas, Pilha Robusta [Pra98]. Essa aplicação usa dois mecanismos alternativos para tolerar falhas de software: blocos de recuperação e N-versões. Três variantes de pilha com diversidade de projeto são utilizadas.

As falhas consideradas nos experimentos foram baseadas na corrupção de valores dos parâmetros de métodos públicos; quanto ao padrão de repetição foram injetadas falhas transientes, intermitentes e permanentes.

Os testes realizados serviram para revelar erros de implementação da aplicação alvo e verificar diferenças no potencial de detecção de erros dos tipos de falhas considerados pela ferramenta.

---

[2] Embora Openc++ 1.2 não permita reflexão em caracterísiticas privadas de um objeto, falhas internas podem ser consideradas para as características protegidas.41

Foram revelados dois erros: um erro na implementação de uma das variantes e um erro na implementação do mecanismo blocos de recuperação. Quanto a eficiência na detecção de erros, nos experimentos realizados verificamos que:

- o momento da injeção de falhas foi um fator muito importante: as falhas injetadas desde o início da execução da aplicação alvo não revelaram o erro de implementação no mecanismo de blocos de recuperação, apenas quando começamos a variar o início da ativação da injeção de falhas esse erro foi descoberto;
- as falhas transientes e intermitentes foram mais eficientes: as falhas permanentes não detectaram o erro de implementação de uma das variantes.

5. Conclusões

O uso da abordagem metaobjeto permite implementar as características de injeção de falhas e monitorização independentemente da funcionalidade da aplicação. Comparando-se a outras abordagens de injeção de falhas por software duas outras vantagens podem ser destacadas. Primeiro, esta abordagem facilita a introdução da instrumentação no código da aplicação: não são inseridas explicitamente instruções de *trap* no código da aplicação e não é necessário executar a aplicação em modo traço. Segundo, esta abordagem simplifica o uso, pois não é preciso executar a aplicação em modo privilegiado, nem é necessário utilizar as características especiais do hardware as quais não são disponíveis para a maioria dos usuários.

Os experimentos realizados com a ferramenta demonstraram sua eficiência na detecção de erros de implementação da aplicação alvo. A aplicação já havia sido avaliada utilizando teste de análise de mutantes, mas os problemas de implementação não haviam sido revelados.

Nosso próximo passo é tentar realizar testes em um aplicação orientada a objetos reflexiva. Neste caso, os mecanismos de tolerância a falhas são implementados no meta-nível e as características de injeção e monitorização deverão ser incorporadas a um meta-meta-nível.

Referências

[CMS95] Carreira, J.; Madeira, H.; Silva, J.G. "Xception: a software fault injection and monitoring in processor functional units". *5th IFIP International Working Conference on Dependable Computing for Critical Applications*, Urbana-Champaign, Illinois, USA, 1995, pp. 135 -149

[Chi93] Chiba S., *"Open-C++ Release 1.2 Programmer's Guide"*, Technical Report n° 93-3, Dept. Information Science, University of Tokyo, 1993.

[Hof89] Hoffman D., "Hardware Testing and Software IC's", *Proc. Pacific NW Software Quality Conference*. Portland, Oregon, sept 1989.

[HTI97] Hsueh, M.C.; Tsai, T.K.; Iyer, R.K."Fault Injection Techniques and Tools". *IEEE Computer*,April/1997, pp.75-82.

[KKA92] Kanawati, N.; Kanawati, G.; Abraham, J. "FERRARI: A Tool for the Validation of System Dependability Properties". *Proc. FTCS-22*, IEEE CS Press, Los Alamitos, Calif., 1992, pp. 336-344.

[Lis97] Lisboa, M.L.B. *"Computational Reflection in the Object Model"*. Tutorial presented at II Brazilian Symposium of Programming Languages, Campinas, SP, Brazil, 1997, 53 pages.

[Mae87] P. Maes, "Concepts and Experiments in Computaional Reflection". *Proc. OOPSLA'87*, p. 147-155, 1987.

[Pra98] D. Piubeli Prado. *"Implementação de Sistemas Tolerantes a Falhas Usando Programaçao Orientada a Objetos"*. Dissertação de Mestrado, Instituto de Computação, Universidade Estadual de Campinas, jan/98, 88 páginas.