Integrando Injeção de Falhas ao Perfil UML 2.0 de Testes

Júlio Gerchman[1], **Taisy S. Weber**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{juliog,taisy}@inf.ufrgs.br

Abstract. *The UML 2.0 Test Profile (U2TP) is a UML extension which allows the specification, visualization and development of test artifacts for system verification and validation. However, the profile does not offer any feature which allows direct integration and description of test scenarios using fault injection, an efficient technique for testing fault tolerance mechanisms. This work proposes an extension of U2TP to allow the description of scenarios and elements required for using fault injection techniques on systems validation.*

Resumo. *O Perfil UML 2.0 de Testes (U2TP) é uma extensão de UML que permite a especificação, visualização e construção de artefatos de testes usados na verificação e validação de um sistema. No entanto, esse perfil não oferece mecanismos para integração direta de injeção de falhas, técnica eficiente para o teste de mecanismos de tolerância a falhas. Este trabalho propõe uma extensão de U2TP, permitindo descrever cenários e elementos necessários para o uso de técnicas de injeção de falhas.*

1. Introdução

O uso de sistemas de computação em aplicações críticas faz com que seus clientes exijam níveis de dependabilidade cada vez maiores. Dependabilidade pode ser definida como o grau de confiança que se pode depositar em um sistema computacional, representando a soma de atributos como confiabilidade, disponibilidade, integridade e segurança funcional (*safety*) [Avizienis et al. 2004]. Para garantir os níveis exigidos, mecanismos de tolerância a falhas são implementados nos sistemas, possibilitando a continuidade de sua operação mesmo na presença de falhas.

O teste dos mecanismos de tolerância a falhas é fundamental, assegurando que o produto vai de encontro aos requisitos estabelecidos. No entanto, a adição desses mecanismos aumenta a complexidade dos testes necessários para a validação do sistema: não apenas os aspectos funcionais devem ser avaliados, mas também os requisitos de dependabilidade devem ser validados.

Injeção de falhas é uma técnica experimental de validação de sistemas que se mostra eficiente em cenários de teste de sistemas com requisitos de dependabilidade. Nesta técnica, falhas são inseridas artificialmente no ambiente de execução do teste, ao mesmo tempo em que o sistema é monitorado e avaliado. Seu uso possibilita um conhecimento mais profundo do comportamento do sistema na ocorrência de falhas.

Integrar as técnicas de tolerância a falhas na metodologia usada para o teste de sistemas permite um melhor entendimento dos mecanismos pela equipe de garantia de

qualidade (*quality assurance*, QA) e dos requisitos de qualidade pela equipe de desenvolvimento. Para suportar as metodologias de teste e facilitar a comunicação entre as diversas equipes, o *Object Management Group* (OMG) desenvolveu o Perfil UML 2.0 de Testes (*UML 2.0 Testing Profile*, U2TP). O U2TP é uma extensão da linguagem UML para descrição tanto de aspectos estáticos (arquitetura) como dinâmicos (comportamento) do teste de sistemas.

No entanto, U2TP não oferece elementos que permitam uma integração direta de técnicas de injeção de falhas em testes descritos com essa linguagem. Tentativas de descrição de experimentos que usem injeção de falhas usando elementos comuns de UML e U2TP podem esbarrar na falta de um padrão, gerando diagramas, documentos e artefatos que podem não ser facilmente reaproveitáveis devido ao uso de notações ou definições diferentes.

Este trabalho propõe uma extensão do Perfil UML 2.0 de Testes para integrar a descrição de testes que usem técnicas de injeção de falhas. A extensão permite especificar aspectos estáticos do ambiente, como a configuração das cargas de falha necessárias para os experimentos, bem como aspectos dinâmicos, como suas condições de ativação. A extensão tem como objetivos uma fácil integração com metodologias e com ferramentas existentes.

A Seção 2 apresenta características da técnica de injeção de falhas e iniciativas existentes para descrever ferramentas para este fim. A Seção 3 discute questões de modelagem de testes. A extensão de U2TP para injeção de falhas é apresentada na Seção 4, e um exemplo de seu uso é encontrado na Seção 5. Considerações finais e discussões sobre trabalhos futuros são apresentadas na Seção 6.

2. Injeção de falhas

Adiar a validação dos mecanismos de tolerância a falhas implementados em um sistema pode ser desastroso. O teste desses mecanismos é necessário para assegurar os níveis de dependabilidade oferecidos por um sistema, determinando a cobertura dos mecanismos e avaliando sua eficiência em um ambiente próximo do real. Uma técnica para realizar esse teste é a injeção de falhas, uma técnica experimental de validação de sistemas que tem como foco principal os mecanismos de tolerância a falhas.

A injeção de falhas pode ser usada para atingir vários objetivos, como identificar gargalos de dependabilidade, estudar o comportamento do sistema em um cenário de falhas, determinar a cobertura dos mecanismos de detecção e recuperação de falhas e avaliar sua eficiência e desempenho [Hsueh et al. 1997]. Em um experimento é adotado um modelo de falhas, uma coleção de atributos e de regras que governam a interação entre os componentes que falharam. Este tipo de modelo permite planejar os mecanismos de tolerância a falhas e descrever os cenários a serem usados durante o teste.

Existem diferentes formas de aplicação da técnica: simulação (prevendo o comportamento através da simulação de modelos do sistema), hardware (usando equipamentos como sondas para alterar fisicamente o ambiente) e software (emulando a ocorrência de falhas através da execução de código). A injeção por software é a mais comum, pois evita certos problemas da injeção por hardware, como o alto custo e a possibilidade de destruição de componentes, e usa protótipos funcionais, ao contrário da injeção em simulação, o que permite uma maior aproximação do comportamento real do sistema.

A ferramenta usada para controlar o experimento, o injetor de falhas, pode atuar através de três técnicas. Na injeção ativa, um processo em paralelo, com as mesmas permissões de acesso do sistema alvo, altera o estado o ambiente de execução quando necessário. A injeção por alteração de fluxo de controle aproveita ganchos oferecidos pelo ambiente de execução, monitorando o alvo e introduzindo falhas quando necessário. Existe também a injeção por alteração de código, onde uma versão especial do sistema é construída, contendo código especial para emular falhas em pontos específicos.

Devido à grande variedade de níveis de abstração, tipos de sistema e ambientes de execução, não existem ferramentas genéricas de injeção de falhas. As ferramentas normalmente são voltadas a um ambiente ou plataforma específica, injetando falhas de uma certa categoria, como, por exemplo, comunicação, memória ou processador. Apesar disso, os ambientes de injeção de falhas apresentam componentes similares [Hsueh et al. 1997]. Um gerador de carga de trabalho produz as entradas para o sistema, enquanto que um injetor de falhas altera o ambiente seguindo regras estabelecidas em uma biblioteca de falhas. Um monitor mantém um registro do comportamento do sistema, passado posteriormente para um analisador de dados. Todos esses componentes são gerenciados por um controlador do experimento.

Para facilitar a construção destas ferramentas, Leme, Martins e Rubira propõem um sistema de padrões para injetores de falhas, um conjunto de recursos e elementos comuns que podem ser usados para acelerar e simplificar seu desenvolvimento [Leme et al. 2001]. Um dos padrões apresentados no trabalho é um padrão arquitetural, que sugere uma estrutura geral para uma ferramenta. Ele divide o problema em cinco subsistemas distintos: *activator* (inicia a execução do sistema alvo), *injector* (introduz falhas no ambiente), *monitor* (verifica as condições do sistema alvo), *controller* (gerencia todos os subsistemas) e *user interface* (permite ao usuário configurar o experimento e recuperar resultados). Os padrões não definem nem um modelo de falhas nem uma linguagem de programação específica.

O padrão arquitetural proposto descreve a estrutura da ferramenta; porém, não se propõe a descrever seus aspectos dinâmicos, como condições de ativação de falhas. A descrição do uso da ferramenta e sua integração com o resto de um plano de testes não faz parte dos objetivos do sistema de padrões proposto. É possível encontrar elementos para especificação de aspectos dinâmicos de um teste no Perfil UML 2.0 de Testes; no entanto, esse perfil não oferece diretamente suporte para injeção de falhas.

3. Modelagem de testes

O teste de um sistema depende da construção de um modelo que o descreva, permitindo a compreensão do comportamento desejado, bem como suas entradas e saídas esperadas. Capturar, formalizar e descrever os modelos usados permite o compartilhamento e reuso destes, facilitando a comunicação entre membros da equipe de garantia de qualidade e permitindo um contínuo melhoramento dos planos de teste para o sistema. A modelagem dos testes, juntamente com sua documentação, permite uma melhor interação entre os desenvolvedores e os engenheiros de teste ao estabelecer um modelo e uma linguagem comum.

Caso metodologias de engenharia de sistemas dirigida a modelos sejam usadas, torna-se ainda mais importante a descrição formal dos testes aplicados. A engenharia

dirigida a modelos é uma abordagem para diminuir a complexidade do projeto de sistemas, separando a lógica de negócio da lógica de aplicação da plataforma usada para sua implementação [Soley 2000]. Modelos independentes de plataforma descrevem a lógica de negócio usando linguagens específicas do domínio da aplicação, como serviços financeiros ou *data warehousing*, aproximando as fases de análise e projeto de especialistas dessas áreas. Ferramentas de transformação podem verificar os modelos e sintetizar artefatos mais próximos à tecnologia usada na implementação do sistema, como descritores XML ou mesmo código-fonte. A existência de modelos de testes permite sua integração com os demais artefatos usados no restante do sistema.

O *Object Management Group* (OMG) lidera a iniciativa mais conhecida na área, a *Model-Driven Architecture* (MDA). O OMG, grupo responsável pela criação e manutenção de padrões como UML (*Universal Modeling Language*), define uma série de linguagens e técnicas para o desenvolvimento usando MDA. Um dos enfoques usados pelo grupo é a especificação de perfis UML (*UML profiles*), extensões para a linguagem UML padrão que adicionam elementos específicos a domínios de aplicação. Seu objetivo é facilitar a análise de sistemas e descrição de modelos ao disponibilizar um conjunto de elementos que tragam os especialistas do domínio mais próximo ao seu desenvolvimento.

A área de testes representava um problema para MDA porque existiam linguagens de descrição distintas para diferentes domínios e níveis de teste. Por exemplo, a área de telecomunicações usa a linguagem TTCN-3 [Schieferdecker et al. 2003]. Além disso, era difícil encontrar uma linguagem que pudesse descrever testes em seus diferentes níveis, desde unitário até integração.

Observando a falta de uma linguagem padrão para a descrição de testes, o OMG criou o Perfil UML 2.0 de Testes (*UML 2.0 Testing Profile*, U2TP). U2TP define uma linguagem para projetar, visualizar, especificar, analisar, construir e documentar os artefatos de sistemas de teste [Object Management Group 2005]. Destinada ao teste em vários domínios de aplicação de sistemas construídos com as principais tecnologias orientadas a objeto, o perfil foi especificado com foco em reuso e em integração com modelos e elementos UML pré-existentes.

U2TP é organizado em quatro grupos de conceitos, cada um tratando de aspectos específicos e oferecendo diferentes elementos. Conceitos de *arquitetura de teste* descrevem os aspectos estáticos usados na verificação de um sistema, permitindo especificar os componentes presentes em um teste. Esses componentes incluem o sistema alvo (*system under test*, SUT), coleções de casos de testes, árbitros que determinariam um veredito de um experimento (como sucesso ou falha) e escalonadores para controlar a execução. Os aspectos dinâmicos, por sua vez, são descritos usando conceitos de *comportamento de teste*. Esses aspectos incluem a definição de vereditos, invocação de ações e de testes e monitoramento através de *logs*.

Valores de entrada e saída podem ser especificados através de conceitos de *dados de teste*, que estendem elementos de UML padrão incluindo elementos como regras para geração e seleção de dados e coringas. O último grupo de conceitos, *temporização de testes*, permite especificar restrições de tempo em um teste e oferece mecanismos de sincronização, na forma de definições de fusos horários.

Apesar dos diferentes elementos oferecidos, o Perfil UML 2.0 de Testes não ofe-

rece características para, de maneira padronizada, descrever um teste que use técnicas de injeção de falhas. Caso seja necessário usar essas técnicas, a equipe de testes deveria estabelecer individualmente padrões e notações, dificultando seu reuso por outras equipes ou projetos.

4. Descrevendo injeção de falhas em UML

Usando apenas UML padrão ou o perfil de testes, a descrição de uma atividade de verificação de sistemas deve ser realizada usando apenas elementos genéricos, sem seguir um padrão estabelecido. Isto introduz uma série de problemas de compatibilidade. A integração entre equipes diferentes, como as equipes de teste e de desenvolvimento de um projeto, é dificultada pela possibilidade de uso de linguagens e notações diferentes. Além disso, o uso de ferramentas é dificultado, necessitando a personalização e impossibilitando o uso de componentes prontos.

O uso de padrões de projeto para a descrição de injetores de falhas, como descrito por Leme, Martins e Rubira, resolve apenas parte do problema da descrição de experimentos de injeção de falhas. Enquanto que aspectos estáticos, como estrutura da ferramenta, componentes do ambiente e associações entre as implementações de injetores e sensores e o sistema sob teste são satisfatoriamente descritos, aspectos dinâmicos, como os parâmetros de configuração e condições de ativação de falhas, não fazem parte do objetivo proposto.

Como o Perfil UML 2.0 de Testes é um padrão oficial da OMG e oferece elementos para descrição de demais conceitos de teste, a abordagem natural é estender o perfil, integrando funcionalidades para a especificação de testes que usem técnicas de injeção de falhas.

Alguns objetivos gerais foram seguidos durante o desenvolvimento da extensão de U2TP para injeção de falhas proposta neste artigo:

- Uma fácil *integração com modelos de testes existentes*. Nenhum elemento de U2TP foi retirado ou teve suas regras de formação alteradas: apenas novos elementos para injeção de falhas foram adicionados. Esses elementos são, em sua maioria, *estereótipos*. Um estereótipo marca um elemento existente em um modelo, como uma classe, como uma extensão ou classificação de algum outro.
- O *reuso de ferramentas* seria possibilitado pelo uso de mecanismos de extensão UML que pudessem ser facilmente incorporados em sistemas CASE existentes. Estereótipos são bem suportados em ferramentas existentes, e seu uso normalmente não necessita a instalação de extensões ou de outros mecanismos para aumentar o número de elementos oferecidos ao usuário.
- A *integração com outras iniciativas* de modelagem de sistemas de injeção a falhas, como o sistema de padrões para injetores descrito por Leme, Martins e Rubira. Os elementos oferecidos pela extensão deveriam ser genéricos o suficiente para serem incorporados facilmente em iniciativas existentes.
- Facilitar a *transformação* do modelo de testes de forma a produzir artefatos, como arquivos de configuração, para uso em ferramentas de injeção de falhas existentes.

4.1. Arquitetura geral

Os elementos da extensão de U2TP para injeção de falhas estão contidos em um pacote UML, representado na Figura 1.

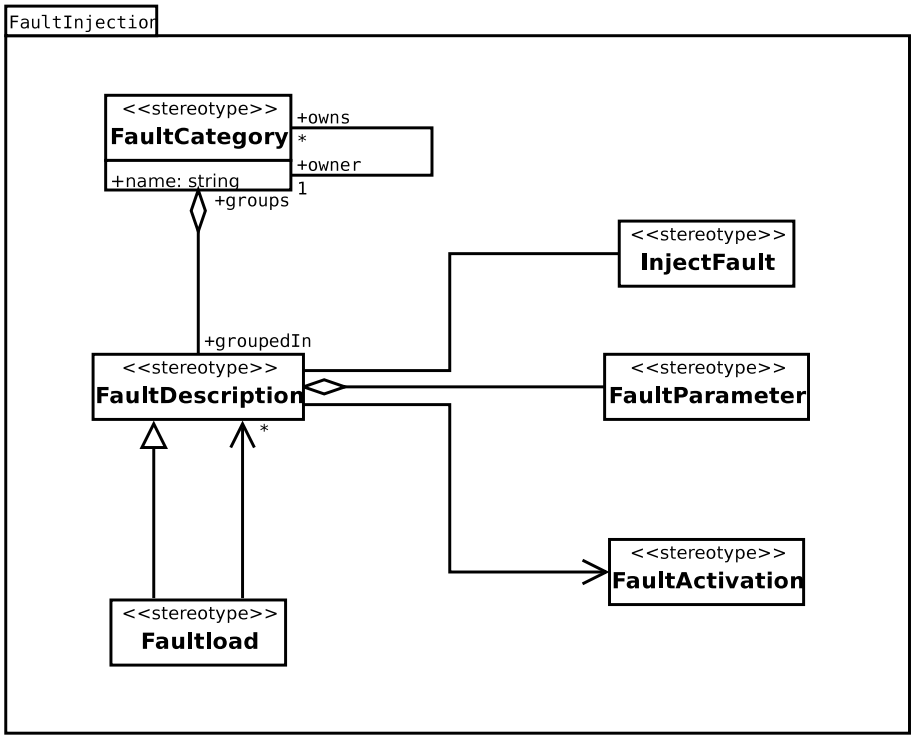

Figura 1. Arquitetura geral da extensão

A extensão é composta por seis estereótipos, com três funções compreendidas:

- A *descrição* de uma falha a ser injetada durante o teste de um sistema é indicada pelos estereótipos «FaultDescription» e «FaultParameter». Classes marcadas com esses estereótipos representam a configuração de um tipo de falha. Modelos que compreendam falhas de comunicação, por exemplo, definiriam classes que representem perda, duplicação, corrupção ou atraso de pacotes.
- A *ativação* de uma falha é indicada pelos estereótipos «FaultActivation» e «InjectFault». O primeiro estereótipo é aplicado a uma classe que descreve as condições de ativação de uma determinada falha. Por exemplo, em um modelo que compreenda diferentes distribuições probabilísticas, a distribuição escolhida pode ser colocada como um atributo da classe que representa uma condição de ativação. O segundo estereótipo é aplicado em uma relação de dependência que associe uma descrição de falha a um evento de interesse, como uma mensagem.
- Falhas podem ser *agrupadas* usando os estereótipos «Faultload» e «FaultCategory». O primeiro pode ser introduzido em um modelo para montar um cenário de falhas, agrupando instâncias de «FaultDescription». «Faultload» foi modelado como uma especialização de «FaultDescription», o que permite indicar a ocorrência de múltiplas falhas da mesma forma que uma falha individual. O segundo é útil para documentar injetores, agrupando categorias abrangentes como falhas de comunicação.

A modelagem de um cenário de testes não usaria diretamente os estereótipos citados acima. Classes específicas para a ferramenta de injeção usada são definidas, contendo atributos que permitam descrever seu comportamento. Por exemplo, no caso de um injetor que atue alterando dados na chamada de um método, uma classe BitFlipFault,

marcada com o estereótipo «FaultDescription», representaria a falha a ser injetada pela ferramenta. Nesse mesmo exemplo, caso seja interessante testar diferentes configurações de falhas, objetos diferentes da classe BitFlipFault, cada um com parâmetros diferentes, podem ser associados a testes específicos.

A arquitetura apresentada foi inspirada no perfil UML para especificação de qualidade de serviço (QoS) e mecanismos de tolerância a falhas (*UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms*) [Object Management Group 2006]. Assim como este trabalho de extensão de U2TP para injeção de falhas, o perfil para QoS permite associar características (no caso, valores e restrições) a elementos de um modelo, como classes, mensagens e mesmo associações. A relação entre «FaultDescription» e «FaultParameter» é similar à relação mostrada no diagrama da metaclasse QoSCharacteristic, que também especifica características quantificáveis de serviços (como, por exemplo, *throughput* em uma rede) independentemente dos elementos que são qualificados.

4.2. Descrição de uma falha

Uma ferramenta de injeção oferece ao seu usuário, normalmente um engenheiro de testes, diversas opções de falhas. As ferramentas são voltadas a categorias de falhas, como comunicação através de rede. Cada uma das falhas, no entanto, pode apresentar configurações diferentes de forma a alterar seu comportamento específico. Na extensão de U2TP para injeção de falhas, elas são representadas por classes anotadas com o estereótipo «FaultDescription». Essas classes podem ter associados parâmetros marcados com o estereótipo «FaultParameter»

As falhas descritas podem ser especificadas independentemente do elemento do sistema a que estiverem associadas: isto é, não têm necessariamente uma associação permanente com uma mensagem ou uma classe. É possível descrever completamente uma falha e associar várias de suas instâncias a elementos diferentes: por exemplo, uma mesma falha de paridade de memória pode ser aplicada em diferentes posições, ou mesmo diferentes componentes de um sistema (como um pente de memória e um *buffer* em alguma placa de entrada/saída).

Ao modelar um experimento de injeção de falhas usando a extensão proposta para U2TP, classes que representem as falhas oferecidas pela ferramenta de injeção devem ser descritas. Essas classes devem conter os atributos necessários para descrever seu comportamento durante o experimento. Desta forma, podem ser instanciadas durante a execução do experimento, dando origem a artefatos como arquivos de configuração a serem passados para a ferramenta.

4.3. Ativação de uma falha

É desejável que uma ferramenta de injeção de falhas comece a atuar a partir de um momento posterior ao início dos testes. Por exemplo, um teste pode verificar o funcionamento de mecanismos de tolerância a erros de memória a partir da metade de uma computação. Por outro lado, a freqüência de ocorrência pode não ser constante: uma falha pode ser classificada como permanente, transiente ou intermitente [Carreira and Silva 1998]. No caso de falhas que não sejam permanentes, devem ainda ser levadas em conta considerações sobre sua distribuição probabilística em relação a algum parâmetro como tempo decorrido.

No desenvolvimento da extensão de U2TP para injeção de falhas, os elementos que descrevem as condições de ativação foram modelados separadamente dos elementos que descrevem outros parâmetros de uma falha. Estes elementos são marcados com o estereótipo «FaultActivation». Esta decisão foi tomada pensando no reuso de elementos em um diagrama. Um exemplo desse caso é a ativação de várias falhas diferentes de comunicação apenas a partir da transmissão de uma quantidade específica de *bytes*, o que pode ser útil para simular problemas na comunicação de um cabeçalho correto mas um *payload* problemático. Neste caso, apenas um objeto seria instanciado no modelo, indicando a ativação de falhas a partir de um número de *bytes*; os vários objetos representando falhas seriam associadas a este único objeto.

A associação entre uma falha e o elemento onde deve atuar durante um experimento é uma dependência UML anotada com o estereótipo «InjectFault». Essa associação de dependência liga um elemento de modelagem, como mensagens entre objetos, a uma descrição de falha, objeto anotado com o estereótipo «FaultDescription». «InjectFault» é inspirado no elemento de dependência «QoSContract», parte do perfil UML para modelagem de qualidade de serviço e responsável por associar a um serviço um certo nível de qualidade [Object Management Group 2006].

4.4. Organizando um cenário de falhas

Um cenário de falhas descreve o ambiente de testes onde a aplicação será executada, especificando cada falha e seus respectivos parâmetros de configuração. Esse cenário de falhas atua como uma carga de falhas (*faultload*) para o teste.

Ferramentas de injeção usam diferentes formas de especificar uma carga de falhas. Injetores como FIONA usam arquivos de configuração, descrevendo cada uma das falhas e suas respectivas condições de ativação [Jacques-Silva et al. 2006]. Uma forma diferente é usada na ferramenta FIRMI, onde uma classe Java cria objetos representando as falhas [Vacaro and Weber 2006]. Apesar disso, ambos apresentam o conceito de configuração da carga de falhas.

Na extensão de U2TP decidiu-se criar o estereótipo «Faultload», marcando uma classe como um agrupamento de diferentes classes «FaultDescription». Desta forma, agrupando diferentes falhas, define-se a carga de falhas com a qual o injetor será alimentado.

Para uma melhor documentação e organização de modelos de injetores, criou-se o elemento «FaultCategory». Categorias de falhas fariam um agrupamento abrangente, como “falhas de comunicação” ou “falhas de hardware”. Este elemento é útil na modelagem de injetores de falhas, de forma a indicar de forma ampla o objetivo da ferramenta.

5. Exemplo: chamada a *web service*

Como demonstração do uso da extensão U2TP de injeção de falhas proposta neste artigo, foi modelado um exemplo de caso de teste de uma aplicação *web* que mostra informações de recursos humanos sobre empregados de uma empresa. Essa aplicação *web* teria uma camada de apresentação (*frontend*) construída usando o *framework* Struts, que usa a plataforma Java 2 Enterprise Edition (J2EE). A camada de apresentação se comunicaria com a camada de negócio (*backend*) através de *web services*, usando SOAP (*Simple Object Access Protocol*) sobre o protocolo HTTP. A camada de negócio é executada em uma

máquina diferente da camada de apresentação. Esta arquitetura de sistemas é muito usada na atualidade, por ser de fácil construção, oferecer bom isolamento entre as camadas que compõem a aplicação e apresentar boa flexibilidade, permitindo inclusive o uso dos *web services* por clientes diferentes da camada de apresentação *web*.

O caso de teste verificaria se as classes responsáveis pela chamada de *web services* reportam corretamente erros para o usuário caso o servidor de aplicações onde a camada de negócios executa não esteja funcionando. Este cenário de falha pode acontecer se o programa servidor for encerrado, seja por um comando de administração ou por um erro, como falta de memória, que cause seu término pelo sistema operacional. Tentativas de chamada de serviços pela camada de apresentação retornariam erros indicando que a porta TCP estaria fechada.

Como injetor de falhas de comunicação foi escolhida a ferramenta FIERCE, destinada à validação de aplicações Java que usem o protocolo TCP/IP [Gerchman and Weber 2006]. Na nomenclatura usada por FIERCE, a falha descrita acima é uma falha de término de processo (*kill*). A ferramenta oferece dois meios diferentes de ativação da falha: por tempo decorrido ou por *bytes* transmitidos a partir da inicialização do injetor.

O primeiro passo para modelar o caso de teste é criar um modelo descrevendo as falhas oferecidas pela ferramenta de injeção, mostrado na Figura 2. O injetor FIERCE permite injetar falhas de colapso de nó (*crash*) e de término de processo (*kill*). A configuração de ambas as falhas exige os endereços e portas de origem e destino, o que justifica criar uma superclasse com os atributos em comum, e duas classes que as representem. Essas classes são anotadas com o estereótipo «FaultDescription». De forma semelhante, as falhas podem ser ativadas pelos mesmos meios (tempo ou *bytes*), representadas por classes anotadas com o estereótipo «FaultActivation».

O caso de teste simula uma requisição de um usuário que deseja recuperar informações sobre um empregado registrado no sistema. A camada de apresentação, para buscar essas informações, precisa contatar o *backend*, passando a identificação do empregado. Esta tarefa é realizada através uma classe utilitária, responsável pela chamada de *web services*.

A falha deve ser injetada no momento da chamada do *web service*. A classe utilitária que realiza a chamada deve disparar uma exceção indicando a indisponibilidade do serviço. A camada de apresentação, ao capturar essa exceção, deve exibir uma mensagem de erro para o usuário.

O caso de teste inicia invocando o método `execute` da classe `GetEmploymentInformationAction`; em uma aplicação construída com o *framework* Struts, ela seria responsável pelas requisições originadas da página *web* correspondente. Essa classe invoca o método `call` da classe utilitária `WebServiceAdapter`, responsável pela chamada de *web services*. `WebServiceAdapter` inicia a requisição SOAP sobre HTTP, mas a ferramenta de injeção *FIERCE* atua neste momento, emulando uma falha de término de processo. Uma exceção do tipo `ServiceException` é disparada, causando um encaminhamento da interface com o usuário para uma página de erro. Esse encaminhamento marca o sucesso do caso de teste, fato indicado pelo ação de validação pass, um elemento

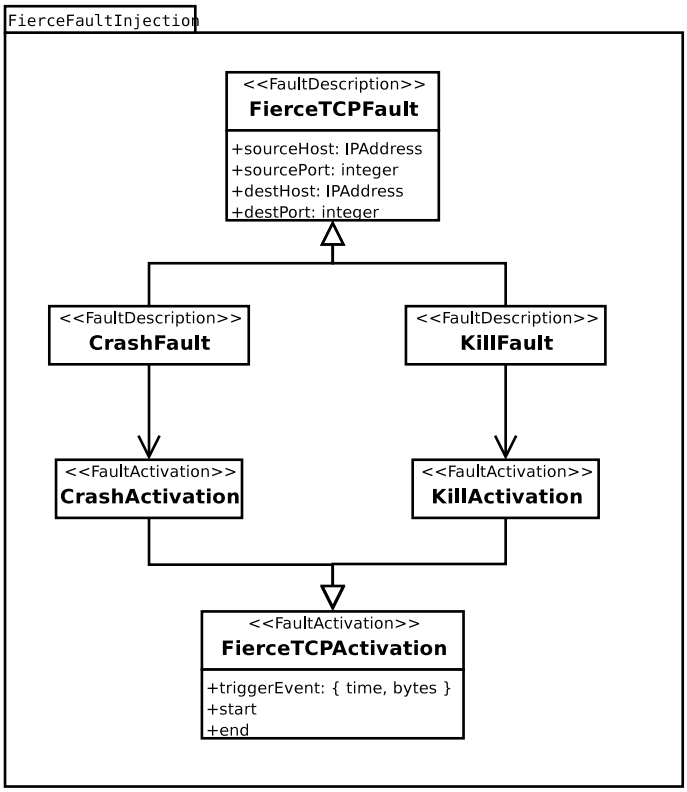

Figura 2. Falhas oferecidas pelo injetor FIERCE

definido por U2TP.

O diagrama de seqüência resultante é mostrado na Figura 3. Neste diagrama, a indicação de injeção de falha é realizada pela relação anotada com «InjectFault» entre a mensagem `getEmploymentInfo` e a descrição de falha `WebContainerCrashed`. Essa descrição de falhas é associada a uma condição de ativação de falha, representada pela classe `WebContainerCrashedActivation`, anotada com o estereótipo «FaultActivation». Neste caso de teste, a falha pode ser ativada desde o primeiro momento da execução, atributos indicados no diagrama.

6. Considerações Finais

O teste de sistemas é uma etapa importante no desenvolvimento de sistemas computacionais; se níveis mais elevados de dependabilidade são exigidos desses sistemas, a validação dos mecanismos de tolerância a falhas torna-se essencial. Injeção de falhas é uma técnica de validação experimental de sistemas flexível e de baixo custo.

O Object Management Group, entidade responsável pelo desenvolvimento da linguagem de modelagem UML, oferece uma extensão para a descrição de testes de sistemas computacionais na forma do Perfil UML 2.0 de Testes (U2TP). U2TP tem como objetivo descrever artefatos usados durante testes, estabelecendo-se como uma linguagem comum entre equipes de desenvolvimento e de testes. O uso de uma linguagem comum possibilita uma maior integração entre as duas equipes e um reuso de esforços de modelagem e projeto de sistemas. No entanto, U2TP não oferece elementos que, de forma uniforme e padronizada, representem os componentes necessários para a descrição de um experi-

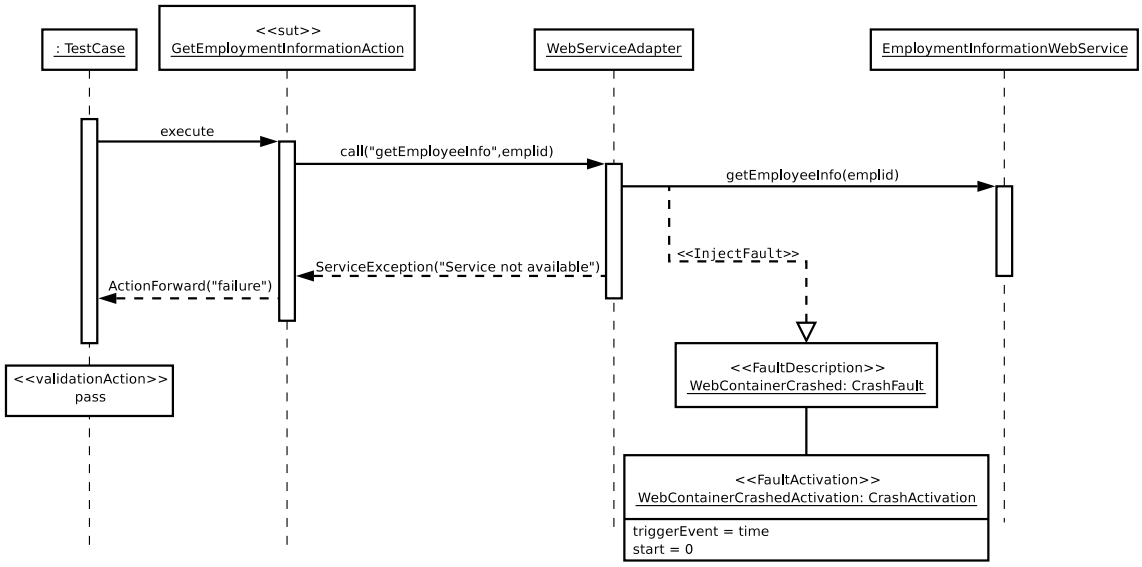

Figura 3. Caso de teste para aplicação *web*

mento de injeção de falhas.

Esse trabalho apresenta uma extensão para o Perfil UML 2.0 de Testes, integrando elementos para a descrição de testes usando técnicas de injeção de falhas. A extensão para U2TP tem como objetivos a fácil integração com ferramentas e metodologias existentes. O uso dos novos elementos oferecidos permite descrever aspectos dinâmicos de um experimento de injeção de falhas, como a especificação de condições de ativação e de associações entre falhas e elementos presentes no modelo do sistema.

O caso de teste mostrado na Seção 5 exemplifica o uso da extensão para U2TP. É demonstrado o uso de uma ferramenta de injeção de falhas de comunicação que atua no protocolo TCP/IP em uma aplicação *web* que realiza chamadas a *web services*.

Trabalhos futuros incluem a formalização da extensão de U2TP usando funcionalidades padrão da OMG como a linguagem OCL, usada para descrição de regras de formação de diagramas. Além disso, a geração automatizada de casos de teste a partir de diagramas de seqüência pode ser explorada.

Referências

Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. E. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, pages 11–33.

Carreira, J. and Silva, J. G. (1998). Why do some (weird) people inject faults? *ACM SIGSOFT Software Engineering Notes*, 23(1):42–43.

Gerchman, J. and Weber, T. S. (2006). Emulando o comportamento de TCP/IP em um ambiente com falhas para teste de aplicações de rede. In *Anais do VII Workshop de Tolerância a Falhas*, volume 1, pages 41–54, Curitiba.

Hsueh, M.-C., Tsai, T. K., and Iyer, R. K. (1997). Fault injection techniques and tools. *IEEE Computer*, 30(4):75–82.

Jacques-Silva, G., Drebes, R. J., Gerchman, J., Trindade, J. M. F., Weber, T. S., and Jansch-Porto, I. (2006). A network-level distributed fault injector for experimental validation of dependable distributed systems. In *Proceedings of the 30th Annual International Computer Software and Applications Conference*, volume 1, pages 421–428, Los Alamitos, California.

Leme, N. G. M., Martins, E., and Rubira, C. M. F. (2001). A software fault injection pattern system. In *Proceedings of the IX Brazilian Symposium on Fault-tolerant Computing*, pages 99–113.

Object Management Group (2005). UML 2.0 Testing Profile. Object Management Group, Inc. document formal/05-07-07.

Object Management Group (2006). UML Profile for modeling Quality of Service and Fault Tolerance characteristics and mechanisms. Object Management Group, Inc. document formal/06-05-02.

Schieferdecker, I., Dai, Z. R., Grabowski, J., and Rennoch, A. (2003). The UML 2.0 Testing Profile and its relation to TTCN-3. In *Lecture Notes in Computer Science: Testing of Communicating Systems*, volume 2644, pages 79–94. Springer.

Soley, R. (2000). Model driven architecture. Object Management Group, Inc. White Paper Draft 3.2.

Vacaro, J. C. and Weber, T. S. (2006). Injeção de falhas na fase de teste de aplicações distribuídas. In *Anais do XX Simpósio Brasileiro de Engenharia de Software*, volume 1, pages 161–176, Florianópolis.