# Geração de cargas de falha para campanhas de injeção de falhas a partir de modelos UML de teste

**Júlio Gerchman**[1], **Cristina C. Menegotto**[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 – Brasil

{juliog,ccmenegotto,taisy}@inf.ufrgs.br

***Resumo.*** *Injeção de falhas é uma técnica experimental eficiente e de baixo custo para o teste dos mecanismos de tolerância a falhas de um sistema alvo. Uma extensão para o Perfil UML 2.0 de Testes (U2TP), usado para a modelagem de artefatos de teste, permite a descrição de artefatos necessários para injeção de falhas. Possibilita também a automação de parte do processo, como a criação de cargas de falha a partir de modelos UML. Este artigo apresenta as vantagens desta extensão. Como prova de conceito, apresenta também a geração de cargas de falha para injetores usados no teste de sistemas baseados em troca de mensagens. Diferentes artefatos podem ser gerados a partir de um mesmo modelo UML, tornando o processo de teste por injeção de falhas mais flexível, documentável e reusável.*

***Abstract.*** *Fault injection is an efficient and low-cost experimental validation technique for fault tolerance mechanism testing. An extension to the UML 2.0 Testing Profile (U2TP), used for modeling test artifacts, allows the description of test artifacts required by this technique. When the extension is used, parts of the test process can be automated, such as the extraction of faultloads from UML models. This paper presents the advantages of using the proposed U2TP extension and presents a tool that automatically generates faultloads for different injectors used to test message-passing systems. Several test artifacts can be generated from the same base UML model, showing the flexibility of the approach and also the possibity to reuse and document test artifacts.*

## 1. Introdução

Atividades de teste de software são essenciais em um processo de desenvolvimento para reduzir a quantidade de defeitos presentes nos produtos [Masiero et al. 2006]. A descrição dos testes de um sistema por meio de modelos torna possível sua documentação e facilita sua análise, permitindo não só a integração com os modelos do restante do sistema mas também o reuso de documentação e de ferramentas nas diversas fases do desenvolvimento. O Perfil UML 2.0 de Testes (*UML 2.0 Testing Profile*, U2TP) é uma extensão da linguagem UML para realizar essa descrição [Object Management Group 2005]. Por meio desse perfil, é possível projetar, visualizar, especificar, analisar, construir e documentar os artefatos usados, abrangendo tanto aspectos estáticos (arquitetura) como dinâmicos (comportamento) de teste de sistemas.

U2TP, entretanto, não oferece elementos que permitam uma integração direta de injeção de falhas nos modelos construídos. Injeção de falhas é uma técnica de

validação experimental na qual falhas são artificialmente introduzidas enquanto o sistema alvo é monitorado, de forma a observar o comportamento do alvo nessas condições e determinar a cobertura dos mecanismos de tolerância a falhas implementados [Hsueh et al. 1997]. Consideramos que a integração dessa técnica a U2TP é essencial para permitir a efetiva aplicação de injeção de falhas como complemento aos procedimentos convencionais de teste. Para realizar essa integração, apresentamos em trabalho anterior [Gerchman and Weber 2007] uma proposta para estender U2TP, que aparece refinada neste artigo. A extensão proposta permite a construção de ferramentas que automatizam a geração de cargas de falha. Este trabalho, além de refinar a proposta, explora as possibilidades, os casos e as ferramentas para tratar dessa automatização.

A modelagem de casos reais, mostrando a geração de cargas de falhas a partir de modelos UML, é usada neste artigo como prova de conceito da extensão proposta. O primeiro caso modela a injeção de falhas no *middleware* de computação em grade OurGrid [Cirne et al. 2006]. Uma das funcionalidades de OurGrid é perceber o colapso de um nodo, não mais disponibilizando-o para o usuário. O injetor de falhas de comunicação escolhido para o teste aceita a especificação da carga de falhas em diferentes formatos que, conforme mostrado neste artigo, podem ser gerados automaticamente a partir do mesmo modelo UML. O segundo caso mostra a possibilidade da geração de carga de falhas para o uso de múltiplos injetores no teste de uma aplicação que usa transmissão de dados via UDP e detecção de falhas via TCP. A prova de conceito permite argumentar a respeito da expressividade e das vantagens do uso da extensão proposta.

A Seção 2 discute a técnica de injeção de falhas. A Seção 3 apresenta conceitos do Perfil UML 2.0 de Testes e descreve a extensão para injeção de falhas. A Seção 4 trata da estratégia usada para a geração automática de cargas de falhas. As etapas para modelagem dos casos usados como prova de conceito são mostradas na Seção 5. Na Seção 6, são mostrados os casos modelados e seus resultados. Na Seção 7, são discutidos trabalhos relacionados. Considerações finais são apresentadas na Seção 8.

## 2. Injeção de falhas

Injeção de falhas é uma técnica experimental de validação. Falhas são introduzidas artificialmente no ambiente de execução de um sistema de forma a analisar o comportamento da aplicação, verificando se está de acordo com suas especificações [Carreira and Silva 1998]. Esta técnica permite introduzir artificialmente falhas de hardware, de software ou de interface entre sistemas.

A *injeção de falhas de software* tem como objetivo avaliar casos de teste, selecionando os mais capazes de distinguir comportamentos válidos de inválidos [Sugeta et al. 2004]. A técnica principal é o *teste de mutantes* [Masiero et al. 2006]: são geradas versões alteradas do programa ou da especificação original, usando operadores que os modificam de forma a emular erros comuns que ocorrem durante seu desenvolvimento. Esse artigo não trata de mutantes, mas enfatiza *injeção de falhas por software*, especificamente injeção de falhas de hardware por software.

Um *injetor de falhas de hardware por software* é uma ferramenta em software que atua sobre um sistema alvo seguindo um cenário de falhas. Neste cenário, são descritos os atributos do ambiente, a carga de trabalho e a carga de falhas. Essa última contém os tipos de falhas a serem injetadas e atributos relacionados como ativação, latência, freqüência,

duração, extensão e distribuição de probabilidade, entre outros. O cenário faz parte do plano de testes da aplicação. O foco destes injetores geralmente é o teste dos mecanismos de tolerância a falhas. Esse teste é necessário para assegurar atributos de dependabilidade impostos ao sistema, determinar a cobertura de falhas dos mecanismos e avaliar sua eficiência [Hsueh et al. 1997].

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 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 alvo, enquanto um injetor de falhas altera o ambiente seguindo a carga de falhas. Um monitor mantém um registro do comportamento do sistema, passado posteriormente para um analisador de dados.

## 3. Extensão de U2TP para injeção de falhas

O teste de um sistema sempre depende da construção de um modelo que o descreva. Capturar, formalizar e descrever os modelos usados permite o seu compartilhamento e reuso e é parte da arquitetura dirigida por modelos (MDA) da OMG. A criação de diagramas que representem os elementos de um teste é realizada usando o Perfil UML 2.0 de Teste (*UML 2.0 Testing Profile*, ou *U2TP*). U2TP define uma linguagem para projetar, visualizar, especificar, analisar, construir e documentar os artefatos de sistemas de teste [Object Management Group 2005]. O perfil pode ser usado isoladamente para a manipulação de artefatos de teste ou em uma maneira integrada com o resto da aplicação, manipulando juntamente artefatos de sistema e de teste.

U2TP é organizado em grupos de conceitos, cada um oferecendo elementos como metaclasses e estereótipos que podem ser usados na modelagem. Cada grupo de conceitos compreende questões como arquitetura do teste, comportamento, dados de entrada e saída e temporização. No entanto, U2TP não oferece conceitos que permitam a descrição direta e padronizada de atividades de teste que usem técnicas de injeção de falhas. Se essa descrição for realizada sem usar uma linguagem padrão, tanto a comunicação entre diferentes equipes e projetos quanto a criação de ferramentas de automação podem ser dificultadas, reintroduzindo problemas que U2TP deveria evitar. A abordagem natural para descrever testes que usem injeção de falhas é a extensão do perfil U2TP.

A arquitetura da extensão 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*, o Perfil QoS) [Object Management Group 2006] devido às similaridades na necessidade de especificar condições de funcionamento para componentes de um sistema e restrições para mensagens trocadas entre seus objetos. O Perfil QoS tem como objetivo especificar a qualidade mínima aceita na interação entre componentes e na execução de um sistema; o perfil para injeção de falhas pode ser visto como o inverso desse objetivo, especificando a "falta de qualidade" que deve existir nos componentes usados durante a execução do sistema sob testes. O Perfil QoS não se propõe a especificar os mecanismos usados para a garantia de qualidade de serviço, mas sim qual o nível desta que deve ser oferecido para o sistema. Este objetivo é similar ao proposto pela extensão para injeção de falhas de U2TP: não

descrever a estrutura e implementação de um injetor de falhas, mas sim a falha que ele deve emular durante uma atividade de teste.

U2TP-FI, a extensão do Perfil UML 2.0 para injeção de falhas proposta, oferece elementos que permitem a um engenheiro de teste descrever atividades de verificação e validação de sistemas usando técnicas de injeção de falhas. A arquitetura geral da extensão pode ser vista na Figura 1.

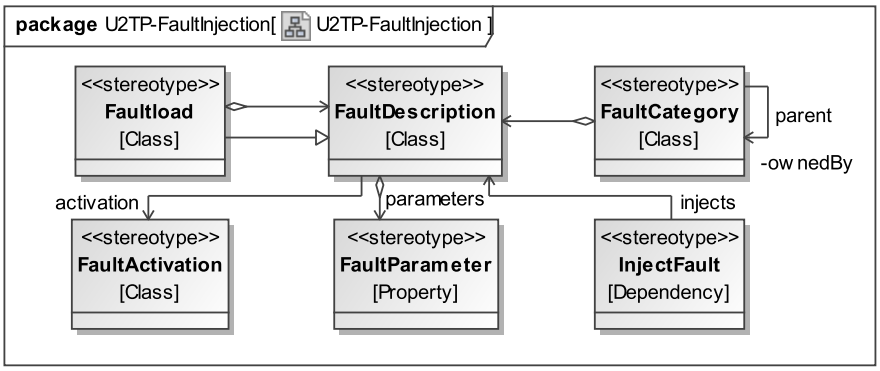


**Figura 1. Arquitetura geral da extensão**

Em U2TP-FI, os diferentes tipos de falhas oferecidos por um injetor são descritos por classes anotadas com o estereótipo «FaultDescription». Parâmetros que permitem variações de seu comportamento são modelados como propriedades associadas e anotadas com o estereótipo «FaultParameter». As condições que ativam uma falha, ou seja, que marcam o início de sua injeção no ambiente de teste, são descritas em classes anotadas com o estereótipo «FaultActivation». Essas condições, por exemplo, podem modelar o tempo de início e a freqüência de ocorrência. Uma descrição de falha, associada a uma ativação de falha, é ligada a um elemento presente no modelo do sistema por meio de uma relação de dependência marcada com o estereótipo «InjectFault». Esta relação indica que a falha será aplicada pelo injetor em um determinado elemento, como um objeto ou uma mensagem enviada entre dois componentes.

Um cenário de falhas é a descrição de um ambiente contendo o sistema em funcionamento, sua carga de trabalho (*workload*) e uma carga de falhas (*faultload*) que atua durante sua execução. Essa carga de falhas é normalmente especificada em um arquivo de configuração fornecido à ferramenta de injeção de falhas ao ser iniciada e contém a descrição das falhas a serem ativadas, bem como seus parâmetros. Em um modelo usando U2TP, as descrições de falhas podem ser agrupadas em uma classe anotada com o estereótipo «FaultLoad», indicando a configuração a ser fornecida à ferramenta. O estereótipo «FaultCategory» é oferecido para documentar ferramentas de injeção de falhas, agrupando falhas de diferentes naturezas em categorias.

## 4. Geração de cargas de falha

A carga de falhas é a descrição das falhas a serem injetadas. A carga de falhas complementa a carga de trabalho (*workload*) aplicada ao sistema alvo em uma campanha de injeção de falhas.

Para gerar a carga de falhas para as campanhas de teste mostradas neste artigo, foi adaptada a ferramenta AndroMDA, um *framework* para a geração de artefatos a partir de

modelos UML, aderindo à abordagem MDA [AndroMDA.org 2007]. Pacotes contendo regras e gabaritos (*templates*) usados para a seleção de elementos de interesse e para a geração de arquivos de saída são chamados cartuchos (*cartridges*).

Um cartucho é um pacote contendo regras para a seleção de elementos de interesse e gabaritos para a geração de arquivos de saída. Para selecionar elementos UML, o principal mecanismo usado por um cartucho é a busca por estereótipos: todos os elementos do modelo que são decorados por um estereótipo específico são agrupados e processados. O processamento desses elementos de interesse é feito por gabaritos, onde atributos e propriedades são usados para preencher lacunas, resultando em arquivos de saída.

Para cada tipo de artefato a ser gerado é criado um cartucho diferente. Por exemplo, é fornecido junto com AndroMDA um cartucho que seleciona classes marcadas com o estereótipo «ValueObject». Para cada uma dessas classes UML, é gerada uma classe Java que segue o padrão de projeto *value object*: cada atributo é transformado em uma variável de instância, acessada por métodos get e set. Em U2TP-FI, a metodologia sugerida para a geração de carga de falhas é a criação de um cartucho para cada injetor, compreendendo o modelo de falhas oferecido. A ferramenta AndroMDA, ao ser invocada, transforma os modelos U2TP-FI em arquivos de configuração para o injetor específico. Assim, o cartucho pode ser reusado entre experimentos com cargas diferentes de falhas, mas que usem as mesmas ferramentas de teste.

A ferramenta CASE usada para a criação de modelos UML foi MagicDraw versão 12.1, desenvolvida pela empresa No Magic, Inc. [No Magic, Inc. 2007]. MagicDraw foi escolhida por ter suporte completo à linguagem UML 2.0 e por ser usada também pela equipe de desenvolvimento de AndroMDA, o que resultou em um ótimo suporte e poucos problemas de compatibilidade. A empresa No Magic oferece uma versão gratuita da ferramenta, denominada *Community Edition*, que tem limitações em relação ao número de elementos e à quantidade de diagramas abertos em uma sessão de modelagem. Como esses limites não seriam atingidos, não mostraram-se como problemas no desenvolvimento dos exemplos de uso de U2TP-FI.

## 5. Metodologia adotada para o desenvolvimento das provas de conceito

U2TP-FI não define uma metodologia ou política de uso. Assim como UML, é apenas uma linguagem a ser usada na representação de modelos de sistemas. No entanto, a metodologia usada no desenvolvimento das provas de conceito pode ser aproveitada por usuários de U2TP-FI como base para a definição de políticas próprias. U2TP-FI visa oferecer elementos para descrever um teste usando técnicas de injeção; a decisão de quais passos usar para realizar essa descrição e integrar os casos de teste ao resto dos modelos existentes no sistema é deixada para o usuário.

As provas de conceito apresentadas neste artigo adotaram uma metodologia em cinco etapas: criação do modelo de falhas, modelagem do caso de teste, criação do cartucho AndroMDA, geração dos artefatos de teste e execução. Cada uma dessas etapas é ligada ou ao injetor de falhas usado ou à campanha de teste em curso. Por exemplo, um modelo de falhas e seu cartucho AndroMDA correspondente podem ser criados apenas uma vez para cada injetor de falhas e reusado em campanhas de teste diferentes. No entanto, se o injetor a ser usado for atualizado, o caso de teste permanece inalterado, mas o cartucho AndroMDA deve ser adaptado à nova ferramenta de teste.

Durante a primeira etapa, a criação do modelo de falhas, os objetivos do teste e a capacidade das ferramentas de injeção são analisados e um modelo UML é criado com classes representando os tipos de falhas de interesse e os tipos de ativação suportados. O resultado dessa etapa é um pacote anotado com o estereótipo «FaultCategory». Este pacote contém as classes de falha que serão referenciadas no modelo. Essas classes apresentam propriedades de interesse para o teste e que são suportadas pelas ferramentas usadas.

A modelagem dos casos de teste é a etapa seguinte. Usando o Perfil UML 2.0 de Testes, os elementos do sistema-alvo, como classes e interfaces, são importadas para o modelo. Os casos de teste são criados como classes com comportamentos associados. Esses comportamentos são especificados por diagramas de seqüência ou de comunicação, mostrando a ordem de mensagens a serem trocadas entre os elementos, determinando as interações entre os componentes do teste. Nesses diagramas, em um primeiro momento, são usados os conceitos U2TP originais, definindo elementos e comportamentos como ações de validação, indicação de dados usados, entre os demais necessários durante uma atividade de teste.

Em um diagrama de classes à parte, são criadas instâncias de descrição e de ativação de falhas. Essas instâncias, objetos associados aos tipos definidos na etapa anterior, têm seus parâmetros definidos. Esses parâmetros definem o comportamento dessas falhas ao serem injetadas durante a campanha de teste.

Nos diagramas de seqüência que definem o comportamento e as ações dos casos de teste, são incluídos elementos que representam as falhas a serem injetadas. Usando relações de dependência decoradas com o estereótipo «InjectFault», eles são ligados aos componentes do teste que indicam os lugares em que as falhas atuarão. Por exemplo, uma falha pode ser ligada a uma classe que representa uma máquina que sofrerá colapso, ou a uma mensagem que terá seus parâmetros corrompidos.

Com os modelos UML representando as falhas e os casos de teste prontos, passa-se à próxima etapa, a criação do cartucho AndroMDA. Um cartucho é criado para cada injetor de teste a ser usado durante a atividade de teste. No desenvolvimento do cartucho, são criados arquivos de descrição que especificam quais são os elementos em que a ferramenta atuará. Um exemplo usado no desenvolvimento dos casos de teste deste artigo é a seleção de todos os elementos UML do tipo instância que estejam anotados com o estereótipo «FaultDescription». Junto a isso são definidos arquivos de gabarito com esqueletos dos artefatos e marcações indicando onde as propriedades das falhas serão preenchidas.

Com o modelo de teste e a ferramenta pronta, a etapa de geração dos artefatos consiste na execução de AndroMDA, fornecendo como entrada os arquivos de modelos UML e os cartuchos desenvolvidos. Essa geração pode ser automatizada por scripts de preparação e compilação (build); por exemplo, nos mesmos scripts que realizam uma compilação de casos de teste que usem o framework JUnit, a ferramenta AndroMDA pode ser invocada, gerando os arquivos de configuração do injetor.

Os arquivos gerados são usados na etapa seguinte, a execução do teste. Para realizá-la, é definido o ambiente experimental de execução, contendo componentes como os injetores, os arquivos de configuração e as estratégias de monitoração e registro.

## 6. Provas de conceito

Para mostrar a expressividade de U2TP-FI na modelagem de testes usando técnicas de injeção de falhas, foram desenvolvidas provas de conceito. Essas provas de conceito envolvem a definição de modelos de teste usando a extensão proposta e a transformação destes para a geração de artefatos que podem ser usados em uma campanha de injeção de falhas.

Nas próximas seções, seguindo a metodologia adotada para o desenvolvimento das provas de conceitos, é apresentada a geração de cargas de falha para duas campanhas de injeção de falhas e uma análise dos resultados alcançados com o uso de U2TP-FI.

### 6.1. Geração de cargas de falha para teste de OurGrid

O sistema sob testes é o *middleware* de computação em grade OurGrid, que coordena a execução de aplicações paralelas *bag-of-tasks* [Cirne et al. 2006]. OurGrid usa o protocolo RMI para comunicação entre seus componentes, tornando possível o uso do injetor FIRMI. Esta ferramenta injeta falhas de colapso, de temporização e resposta em aplicações Java baseadas em RMI, além de oferecer facilidades como a integração com o framework JUnit [Vacaro and Weber 2006].

Um ambiente OurGrid é formado por três componentes. As máquinas disponíveis na grade possuem um *agente*, componente responsável pela execução de tarefas. Esses nodos são agrupados em domínios administrativos organizados por uma máquina denominada *peer*, que atua como porta de entrada, provendo e anunciando os agentes disponíveis para execução. Os *peers* são contatados pela máquina do usuário. Essa máquina do usuário executa o componente MyGrid, responsável pela coordenação e escalonamento de tarefas.

Um exemplo explorado pelos autores de FIRMI foi a verificação da capacidade de OurGrid de perceber a queda de um agente ao listar as máquinas disponíveis. Essa listagem é feita pelo comando `peer status`. Ao ser executado, os *peers* chamam o método remoto `ping` dos agentes conhecidos, e os *hostnames* dos agentes que respondem são retornados ao usuário. FIRMI injeta uma falha de colapso entre um *peer* e um agente da grade em execução na máquina `bentley`. Essa falha só é ativada (ou seja, passa a ser injetada no ambiente) após a segunda chamada a `ping`. Isso permite verificar que o agente está disponível antes da falha e que o *peer* não reporta mais essa disponibilidade depois da ocorrência do colapso. O resultado esperado do teste é a listagem da máquina `bentley` na primeira chamada a `peer status`; uma segunda chamada a esse comando não indicaria mais a máquina.

O primeiro passo foi modelar classes que representam as falhas que são de interesse no teste e que são oferecidas pelas ferramentas, decoradas com o estereótipo «FaultDescription». Da mesma forma, são modeladas classes que representam as condições de ativação, decoradas com «FaultActivation». O diagrama da Figura 2 mostra os elementos criados. Foram representadas falhas de colapso (`CrashFault`) e de temporização (`DelayFault`), ambas especializações de `CommunicationFault`, que tem atributos para indicar os *hostnames* envolvidos. Essa classe tem uma associação a `CommunicationFaultActivation`, que representa as condições que podem ser usadas por FIRMI para ativar uma falha: a partir de um tempo determinado (`TimeActivation`) ou a partir de um número de invocações a um método específico

(`InvocationFault`). O modelo UML foi criado usando a ferramenta MagicDraw UML, versão 12.1.

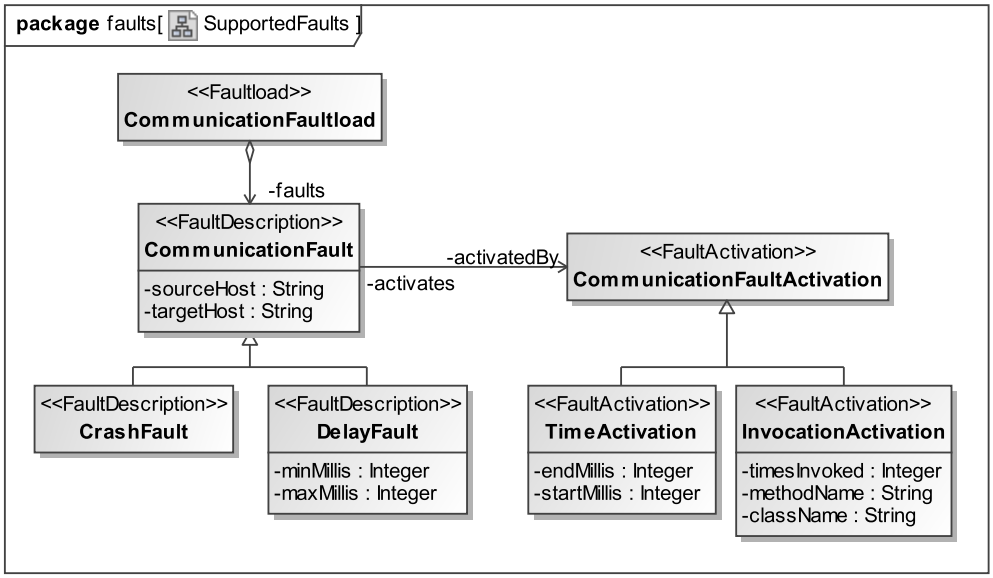


**Figura 2. Falhas oferecidas pelo injetor FIRMI**

O caso de teste foi modelado como um diagrama de seqüência mostrando a interação entre quatro elementos. A classe `BentleyCrashedTestCase` inicia as operações, chamando duas vezes o comando `peer status` de um *peer* OurGrid. O *peer* invoca o método `ping` de um agente representado pelo elemento `bentley`. Essa última mensagem é associada por meio de uma relação de dependência «InjectFault» a uma falha identificada por `bentleyCrashed`. O diagrama pode ser visto na Figura 3.

Um diagrama de classes auxiliar foi usado para especificar os parâmetros das falhas e das ativações usadas no caso de teste. Uma instância de `CrashFault`, chamada `bentleyCrashed`, mostra que a falha ocorre para comunicações que envolvam a máquina de nome `bentley`. As condições de ativação são especificadas em uma instância de `InvocationActivation`, associada a ela. Os atributos indicam que a falha será injetada a partir da segunda invocação do método `ping`, pertencente à classe remota `UserAgentServer`, o agente OurGrid. O diagrama pode ser visto na Figura 4.

Com o modelo pronto, o passo seguinte consiste em processar os diagramas UML e extrair a carga de falhas (*faultload*) para os testes. No caso da ferramenta FIRMI, ela pode ser representada tanto por arquivos de configuração em XML descrevendo as falhas quanto por classes Java que manipulam o estado do injetor e, portanto, da comunicação. Os arquivos XML são de leitura e manipulação mais fácil. Por outro lado, representar o *faultload* por classes Java permite o uso de FIRMI em conjunto com o *framework* de teste JUnit. Dois cartuchos foram criados: `firmi-xml` para a geração de arquivos de configuração XML e `firmi-java` para classes Java. O desenvolvimento de dois cartuchos tem como objetivo mostrar que U2TP possibilita um reuso de modelos ao permitir gerar artefatos diferentes a partir dos mesmos elementos.

Os dois cartuchos estão atrelados ao modelo de falhas especificado na Figura 2. Os modelos UML de entrada são varridos à procura de instâncias (ele-

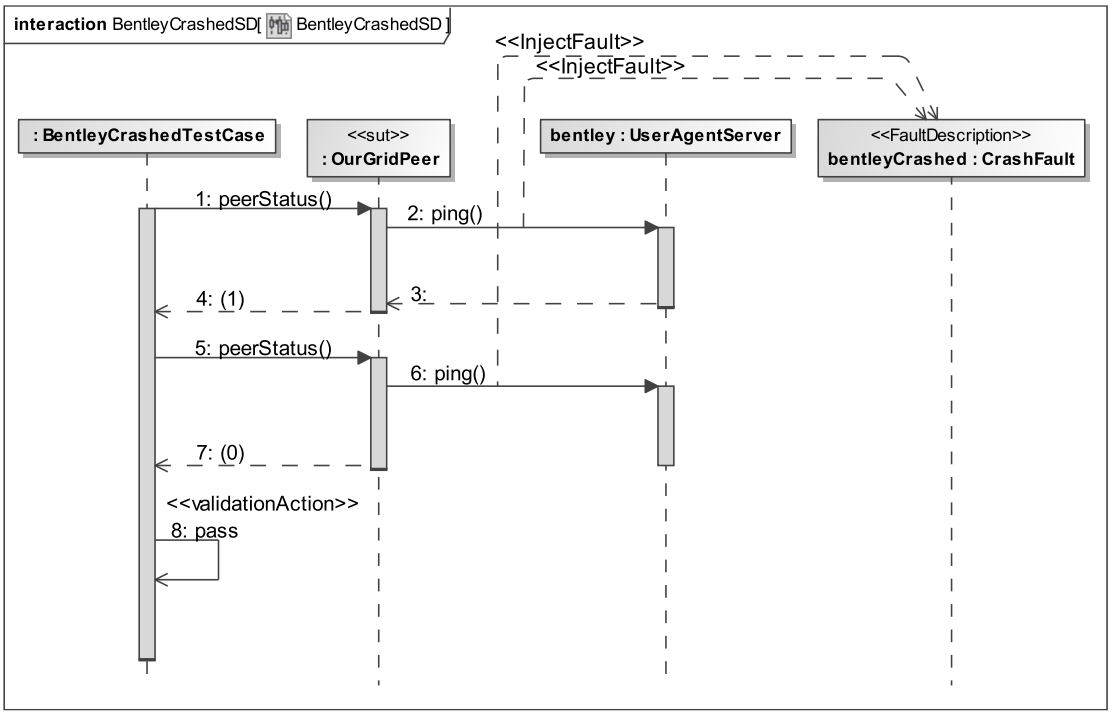


**Figura 3. Caso de teste para avaliação de OurGrid**

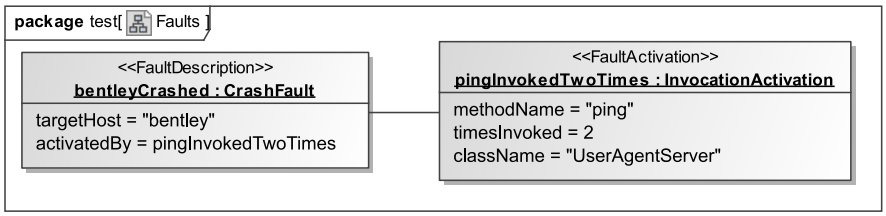


**Figura 4. Especificação das instâncias de falhas**

mentos UML *Instance Specification*). Para cada instância, é verificado se ela corresponde às classes `CrashFault` ou `DelayFault`. Os atributos do elemento são armazenados em variáveis internas, bem como os atributos da instância de `CommunicationFaultActivation` associada.

As variáveis com os atributos extraídos dos modelos UML são usadas por gabaritos que geram os arquivos de saída. No cartucho `firmi-xml`, o gabarito é um esqueleto de arquivo XML; no cartucho `firmi-java`, um esqueleto de classe Java com chamadas para ativar os filtros para comunicação RMI do injetor FIRMI.

AndroMDA foi ativado por um *script* Ant, recebendo como entrada o arquivo com modelo UML e o diretório de saída para os artefatos gerados. A saída do cartucho `firmi-xml` pode ser conferida na Figura 5.

### 6.2. Geração de carga de falhas para múltiplos injetores

O segundo exemplo de aplicação de U2TP-FI é a modelagem de um caso de teste de uma aplicação usando o *framework* de comunicação em grupo JGroups. Essa aplicação foi usada anteriormente para demonstrar o uso do injetor de falhas de comunicação FI-ONA, cujo foco é a validação de aplicações que usam o protocolo UDP para comunicação

```
<XMLFaultload>
  <TriggerGroup>
    <Trigger id="pingInvokedTwoTimes">
      <Counter limit="2">
        <Condition class="UserAgentServer" method="ping"/>
      </Counter>
    </Trigger>
  </TriggerGroup>
  <FaultGroup>
    <CrashFault id="bentleyCrashed" activation="pingInvokedTwoTimes" host="bentley"/>
  </FaultGroup>
</XMLFaultload>
```

**Figura 5. Carga de falhas XML FIRMI para o caso de teste**

[Jacques-Silva et al. 2006].

JGroups permite às aplicações efetuarem *multicast* confiável entre máquinas: ou seja, todas as máquinas participantes de uma sessão de comunicação, garantidamente, recebem todas as mensagens enviadas na mesma ordem em que foram enviadas. As máquinas são chamadas *membros de um grupo*, que constantemente se comunicam para detectar falhas como uma desconexão. JGroups permite ao programador escolher qual a estratégia de transmissão de dados e de detecção de falhas. A aplicação em questão, como era uma demonstração de um injetor de falhas em UDP, usava *multicast* UDP para transmitir os dados entre as máquinas e, como estratégia de detecção de falhas, trocava periodicamente mensagens estilo "ping" também em UDP. A falha em receber uma resposta ao "ping" é interpretada como uma desconexão. Essa estratégia é denominada por JGroups como *detector FD*.

A aplicação de teste foi executada em quatro máquinas: corvette, maverick, buick e bentley. É oferecida ao usuário uma área de desenho; o que é desenhado em uma máquina é transmitido a todos os membros do grupo, que exibem a imagem. No teste realizado, 50 segundos após o início do experimento, uma falha de particionamento de rede era ativada. Nesta falha, as máquinas eram isoladas em duas metades: corvette e maverick não podiam se comunicar com buick e bentley. Para o usuário da aplicação, o que antes era um grupo de quatro máquinas acabava transformado em dois grupos de duas máquinas. A Figura 6, retirada do artigo em que esse teste foi apresentado, exemplifica a injeção desta falha. Uma letra era desenhada em cada máquina do grupo. Nos 50 segundos iniciais, como as quatro máquinas participavam do mesmo grupo, todas receberam as imagens das letras. Após a ativação da falha de particionamento, formaram-se dois grupos. Novamente, uma letra foi desenhada em cada máquina. No entanto, como estavam em grupos diferentes, as letras *E* e *F* foram transmitidas para os dois primeiros membros, e as letras *G* e *H*, para os outros dois.

O framework JGroups oferece outra estratégia de detecção de falhas, conhecida como *detector FDSOCK*, que usa comunicação via protocolo TCP. A transmissão de dados continua com multicast UDP; apenas o mecanismo de detecção de falhas é alterado. Usando FDSOCK, sockets TCP são abertos entre os membros do grupo. Se houver uma desconexão, os sockets para aquele membro são fechados pelo sistema operacional. Esse fechamento é interpretado como uma falha.

Como FIONA apenas injeta falhas em mensagens UDP, não pode ser aplicado para testar FDSOCK. Uma estratégia seria usar em conjunto o injetor FIERCE, que atua sobre

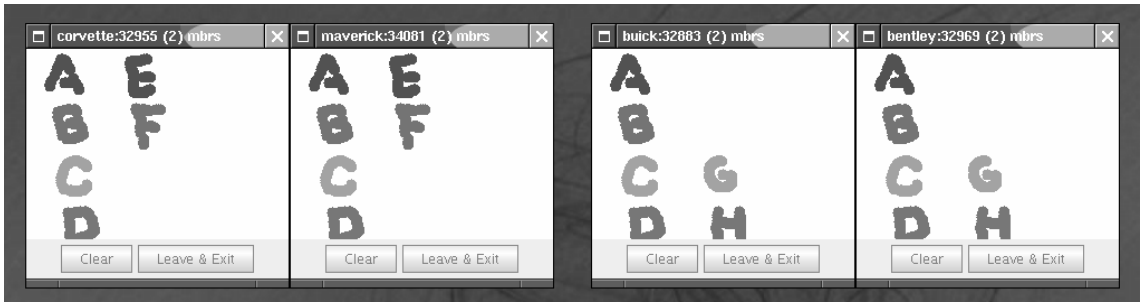


**Figura 6. Aplicação de teste de JGroups com falha particionamento de rede. Fonte: [Jacques-Silva et al. 2006]**

aplicações que usam o protocolo TCP [Gerchman and Weber 2006]. FIONA atuaria sobre a transmissão dos dados, que ocorre usando multicast UDP, enquanto que FIERCE atuaria sobre o detector de falhas de JGroups, que usa TCP. Cabe observar que existem restrições ao uso simultâneo de FIONA e FIERCE relativas à sincronização e temporização. Essas restrições, entretanto, não prejudicam a prova de conceito, pois o objetivo aqui não é validar o resultado das campanhas de injeção de falhas, mas gerar as cargas de falha.

Para a geração da carga de falhas, definiu-se então um modelo de falhas abrangendo as capacidades de ambas as ferramentas de injeção. O modelo final pode ser visto na Figura 7. Das falhas oferecidas por ambos os injetores, apenas a de particionamento de rede foi modelada para este caso de teste. Ela é representada pela classe NetworkPartitionFault, que tem um atributo, nodeList. Para simplificar a implementação, esse atributo é uma *string* que lista as máquinas em cada uma das partições. Dentro de cada partição, as máquinas são separadas por vírgulas; as partições, pelo sinal de dois pontos. Para ativação, foram modeladas as classes TimeActivation, similar à encontrada na seção anterior, e BytesTransferredActivation, que sinaliza o início de injeção de uma falha a partir de um certo número de bytes transmitidos.

Ao contrário de FIONA, a ferramenta FIERCE não suporta de maneira direta uma falha de particionamento de rede. No entanto, ela pode ser facilmente emulada pela criação de falhas de colapso individuais entre os nodos. Isto é, para cada máquina de uma das partições são definidas falhas de colapso tendo como destino as máquinas que são membros das outras partições.

Como parte da modelagem do caso de teste, foi criada a carga de falhas na forma de instâncias de falhas. Essas instâncias apresentam os parâmetros que definem o seu comportamento durante a atividade de validação.

A carga de falhas foi definida no diagrama de classes que pode ser visto na Figura 8. A instância networkPartition modela a falha a ser injetada, especificando uma partição em dois grupos de máquinas. O primeiro grupo contém as máquinas corvette e maverick; o segundo, buick e bentley. Essa instância de falha é associada a afterFiftySeconds, uma instância que indica o início da injeção 50 segundos a partir do início do experimento.

Foram desenvolvidos dois cartuchos para AndroMDA, `fiona` e `fierce`. Os dois cartuchos selecionam no modelo as instâncias do tipo NetworkPartitionFault e as passam para os *templates* de geração de arquivos de configuração. No caso de FIONA, o template é simples, diretamente gerando falhas do tipo UdpNetworkPartitioningFault.

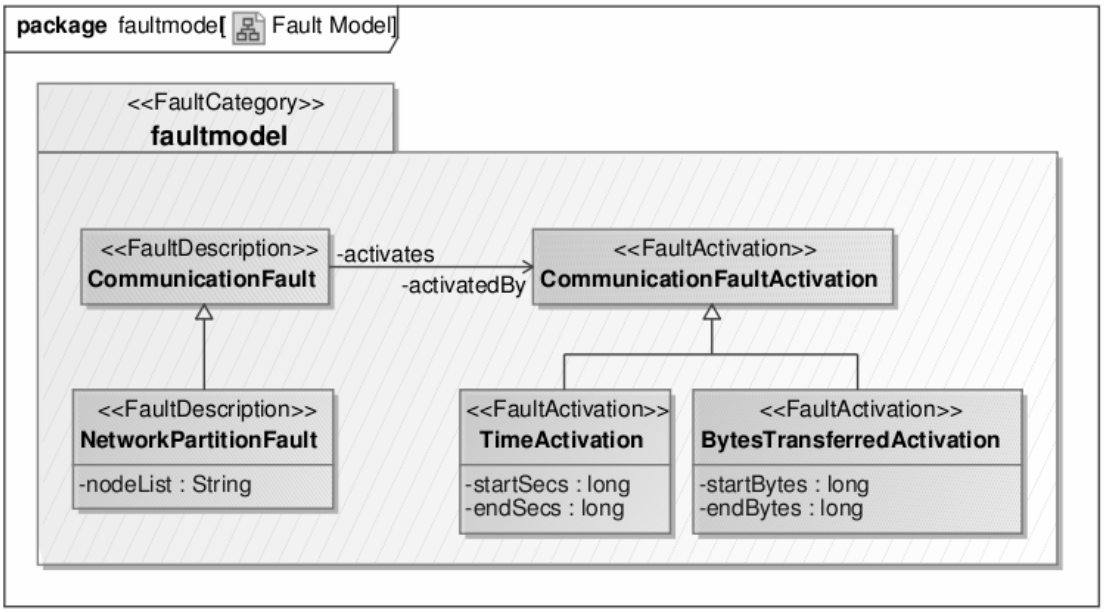


**Figura 7. Modelo de falhas para a aplicação de teste JGroups**

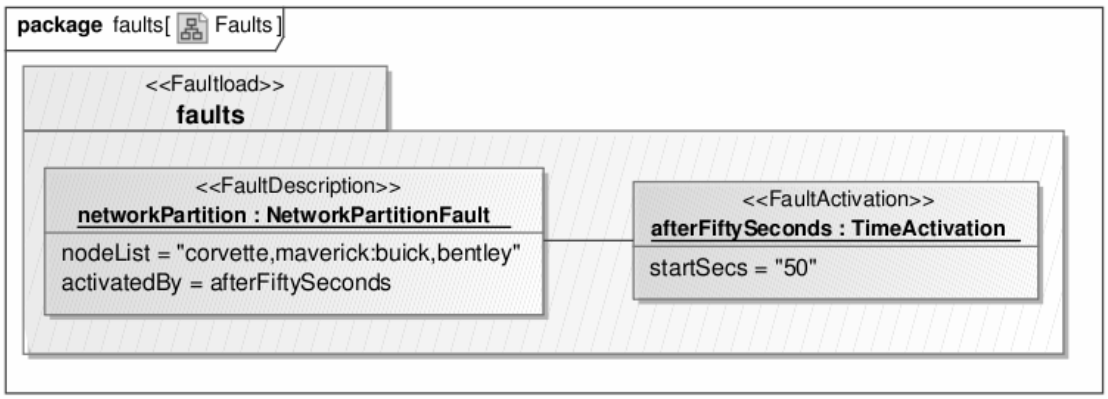


**Figura 8. Carga de falhas para a aplicação de teste JGroups**

Para a geração dos arquivos para FIERCE, foi usada uma pequena classe auxiliar programada em Java. Essa classe apenas faz a quebra da *string* original, que lista os nodos das partições. O *template*, usando essa classe auxiliar, gera as múltiplas falhas do tipo TcpCrashFault necessárias para emular a falha de particionamento.

Finalmente chega-se a etapa de geração dos artefatos de teste. A ferramenta AndroMDA foi invocada nesta etapa para gerar os arquivos necessários a partir dos modelos de teste. No arquivo de configuração de AndroMDA, foram incluídas referências para os dois cartuchos criados, `fiona` e `fierce`. Desta maneira, em apenas uma execução da ferramenta, os dois cartuchos são invocados. Os arquivos de configuração para FIONA e FIERCE podem ser vistos na Figura 9.

Com os arquivos de configuração disponíveis, a campanha de teste usando FIONA e FIERCE poderia ser iniciada.

```
# FIONA: Fault: networkPartition, Activated by: afterFiftySeconds
UdpNetworkPartitioningFault:50:0:corvette,maverick:buick,bentley
```

```
# FIERCE: Fault: networkPartition, Activated by: afterFiftySeconds
TcpCrashFault:50:corvette:*:buick:*
TcpCrashFault:50:corvette:*:bentley:*
TcpCrashFault:50:maverick:*:buick:*
TcpCrashFault:50:maverick:*:bentley:*
TcpCrashFault:50:buick:*:corvette:*
TcpCrashFault:50:buick:*:maverick:*
TcpCrashFault:50:bentley:*:corvette:*
TcpCrashFault:50:bentley:*:maverick:*
```

**Figura 9. Carga de falhas para FIONA e para FIERCE**

### 6.3. Análise dos resultados da aplicação de U2TP-FI

Injeção de falhas é uma técnica de teste, e toda atividade de teste depende de um modelo que a descreva. Esse modelo contém representações do sistema sob testes, da carga de trabalho aplicada, das verificações necessárias e de outros elementos que fazem parte da atividade. No caso de uso de injeção de falhas, elementos pertinentes, como a carga de falhas, também devem ser descritos. A documentação desses modelos de teste usando uma linguagem padrão e visual, como diagramas UML, facilita a comunicação entre as equipes de teste e de desenvolvimento de um sistema, bem como a visualização e análise dele.

Observando o modelo de teste criado, as ferramentas usadas e os artefatos gerados nos dois casos apresentados como prova de conceito nas seções anteriores pode-se argumentar a respeito da expressividade da extensão proposta e das vantagens de sua utilização, que incluem *independência de plataforma*, *documentação de atividades de teste*, *consistência entre diferentes artefatos de teste* e *automação de tarefas repetitivas*.

- *Expressividade da linguagem de descrição de injeção de falhas.* Os estereótipos disponibilizados por U2TP-FI identificam os elementos de injeção de falhas de forma clara em um diagrama U2TP/UML. Os estereótipos «FaultDescription», «FaultActivation» e «FaultParameter» indicam classes que representam falhas e as propriedades que regem seu comportamento. Pacotes decorados com «FaultCategory» e «Faultload» identificam o modelo de falhas adotado e a carga de falhas a ser aplicada em uma atividade de teste. A relação de dependência «InjectFault» mostra claramente o local onde uma falha será injetada. Mostrou-se para experimentos diferentes, usando injetores diferentes, com maneiras diversas de descrição da carga de falhas, que a linguagem consegue facilmente expressar suas particularidades sem qualquer tipo de restrição.
- *Independência de plataforma.* É possível modelar conceitos de injeção de falhas sem associar o modelo de testes a um injetor ou artefato específico, como no caso do segundo exemplo, em que o mesmo modelo de falhas foi usado para a aplicação de duas ferramentas diferentes. O primeiro exemplo também mostrou que diferentes arquivos de configuração podem ser gerados a partir dos mesmos elementos.
- *Documentação de atividades de teste.* Descrever a carga de falhas usando a linguagem UML facilita a transmissão de conhecimento na equipe ao impedir que as

características do teste sejam ocultadas em arquivos de difícil compreensão. Um diagrama de classe é de mais fácil entendimento que arquivos de configuração, carregando mais informações de uma maneira visual.

- *Consistência entre diferentes artefatos de teste.* Os parâmetros de teste ficam centralizados no modelo: não há redundância de informações, o que impede que arquivos de configuração ou outros artefatos de interesse fiquem fora de sincronia. Se for necessária a alteração de algum parâmetro de injeção de falhas, esse parâmetro é alterado diretamente e unicamente no modelo. Os artefatos usados são gerados automaticamente a partir dessa nova versão.
- *Automação de tarefas repetitivas.* Ao centralizar os parâmetros de teste em um modelo UML e delegar a geração dos artefatos necessários à ferramenta MDA usada, a intervenção humana é diminuída, acelerando as tarefas e possibilitando uma maior padronização destas.

## 7. Trabalhos relacionados

Para facilitar a construção de ferramentas de injeção de falhas, 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]. 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 automação de testes a partir de modelos U2TP já foi abordada anteriormente. Uma proposta [Biasi and Becker 2006] é a geração automatizada de *drivers* e *stubs* de teste para o *framework* de teste unitário JUnit a partir de especificações de teste modeladas com U2TP. Entretanto, injeção de falhas e geração automática de cargas de falhas não se encontra no escopo desta proposta.

Diversas ferramentas de injeção de falhas são reportadas na literatura e cada uma delas proporciona diferentes formas de especificar cargas de falhas. Injetores como FIONA [Jacques-Silva et al. 2006] e FIERCE [Gerchman and Weber 2006] usam arquivos de configuração, descrevendo cada uma das falhas e suas respectivas condições de ativação. FIRMI [Vacaro and Weber 2006] possibilita a representação tanto por arquivos de configuração em XML quanto por classes Java que manipulam o estado do injetor. ORCHESTRA [Dawson et al. 1997] é uma ferramenta para validação da implementação de protocolos de comunicação que usa scripts Tcl para a definição das falhas a serem injetadas. FCI (*FAIL Cluster Implementation*) [Hoarau et al. 2007] é uma plataforma distribuída de injeção de falhas em que a descrição de cenários de falhas é feita usando FAIL [Hoarau et al. 2007], uma linguagem abstrata de alto nível desenvolvida para este fim. Com a linguagem FAIL, são especificados cenários de falhas por meio de máquinas de estados e são descritas associações entre cada uma delas e um computador ou grupo de computadores onde a aplicação será executada.

Em geral, as ferramentas representam cargas de falhas de modo descritivo, usando arquivos com indicações das falhas a serem injetadas, ou de modo procedimental, por meio de scripts ou programas que acessam uma API predefinida. Estas duas formas de configuração foram a base para a estratégia de modelagem de falhas de U2TP-FI. Cabe ressaltar que cada ferramenta possui sua própria forma de representação de carga de falhas, o que aumenta o tempo de aprendizagem da ferramenta, dificulta o reuso dessas

cargas de falha e a documentação das campanhas de injeção de falhas inibindo assim, de certa forma, uma maior popularização das próprias ferramentas de injeção de falhas. U2TP-FI, com sua expressividade, independência de ferramenta e automação da geração de cargas de falhas, representa uma contribuição inovadora para a área resolvendo alguns pontos críticos para a maior aceitação de injeção de falhas como estratégia complementar ao teste.

## 8. Considerações Finais

O teste de aplicações é essencial para capturar erros existentes na construção de um sistema computacional e garantir sua qualidade. Injeção de falhas é uma técnica que pode ser usada para a validação experimental de mecanismos de tolerância a falhas destinados a oferecer níveis mais elevados de confiabilidade, disponibilidade e segurança ao sistema. É uma estratégia de teste em que falhas são os estímulos para a atuação dos mecanismos de tolerância a falhas implementados e a cobertura de falhas desses mecanismos uma das principais métricas fornecidas pelo teste. Outras métricas usuais são queda de desempenho sob falhas e tempo médio de recuperação ou reparo. A simples observação do sistema sob falhas permite ao engenheiro de teste determinar pontos fracos do sistema que comprometem sua qualidade ou robustez.

A construção de modelos que descrevam os testes melhora a comunicação entre as equipes e possibilita uma melhor documentação do sistema. A linguagem de descrição padrão é o Perfil UML 2.0 de Testes (U2TP), que permite o projeto, visualização e construção de artefatos de teste. A descrição de atividades de validação que usem técnicas de injeção de falhas é possível usando uma extensão de U2TP. O uso dessa extensão, além de adotar uma linguagem comum para a descrição das atividades de injeção, permite a automação de parte do processo de teste.

Este trabalho apresentou provas de conceito para mostrar a expressividade e demais vantagens da extensão U2TP-FI. Apresentou também uma ferramenta usada para gerar cargas de falhas a partir de um modelo UML criado usando essa extensão. A partir do modelo, cargas de falhas para os injetores de falhas de comunicação FIRMI, FIONA e FIERCE são construídas usando as informações do modelo. Diferentes formatos podem ser gerados, mostrando a expressividade, independência de injetores e flexibilidade da abordagem, entre outros benefícios.

## Referências


AndroMDA.org (2007). What is AndroMDA? Disponível em `http://www.andromda.org/`.

Biasi, L. B. and Becker, K. (2006). Geração automatizada de drivers e stubs de teste para JUnit a partir de especificações U2TP. In *Anais do XX Simpósio Brasileiro de Engenharia de Software*, volume 1, pages 33–48, Florianópolis.

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

Cirne, W., Brasileiro, F., Andrade, N., Costa, L., Andrade, A., Novaes, R., and Mowbray, M. (2006). Labs of the world, unite!!! *Journal of Grid Computing*, 4(3):225–246.

Dawson, S., Jahanian, F., and Mitton, T. (1997). Experiments on six commercial TCP implementations using a software fault injection tool. *Software Practice and Experience*, 27(12):1385–1410.

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.

Gerchman, J. and Weber, T. S. (2007). Integrando injeção de falhas ao Perfil UML 2.0 de Testes. In *Anais do VIII Workshop de Tolerância a Falhas*, pages 87–98, Belém.

Hoarau, W., Tixeuil, S., and Vauchelles, F. (2007). FAIL-FCI: Versatile fault injection. *Future Generation Computer Systems*, 23(7):913–919.

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.

Masiero, P. C., Lemos, O., Cutigi, F., and Maldonado, J. C. (2006). Teste de software orientado a objetos e a aspectos: Teoria e prática. In Breitman, K. and Anido, R., editors, *Atualizações em Informática*, pages 13–71. Editora PUC-Rio: SBC, Rio de Janeiro.

No Magic, Inc. (2007). MagicDraw UML. Disponível em `http://www.magicdraw.com/`.

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.

Sugeta, T., Maldonado, J. C., and Wong, W. E. (2004). Mutation testing applied to validate SDL specifications. In *Proceedings of the 16$^{th}$ TestCom*, pages 193–208.

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.