Ambiente Distribuído de Injeção de Falhas de Comunicação para Teste de Aplicações Java de Rede*

Júlio Gerchman, Gabriela Jacques-Silva†, Roberto Jung Drebes, Taisy Silva Weber

Instituto de Informática – Universidade Federal do Rio Grande do Sul
Caixa Postal 15064 – 90501-970 Porto Alegre, RS

{juliog,gjsilva,drebes,taisy}@inf.ufrgs.br

Resumo. *Sistemas de alta disponibilidade representam um desafio para a meta de garantia de qualidade em seu projeto. Nestes casos, as estratégias implementadas para detecção de erros e recuperação de falhas devem ser criteriosamente testadas. Falhas comuns a ambientes de rede que afetem a troca de mensagens ou provoquem particionamento da rede podem ser emuladas em um ambiente de teste, no qual é executada a aplicação. A observação do comportamento da aplicação sob falhas permite refinar as estratégias de tolerância a falhas implementadas e antecipar a maneira como a aplicação reage a falhas em uma situação real.* FIONA *é uma ferramenta desenvolvida para o teste de aplicações distribuídas de rede escritas em Java e implementadas sob o protocolo UDP. Falhas de comunicação são injetadas através da instrumentação de classes de sistema, carregadas pela máquina virtual Java na inicialização da aplicação, sem alterar seu código.* FIONA *pode ser aplicada em um único nodo da rede ou operar de forma distribuída, facilitando assim o teste de aplicações de larga escala.*

Palavras-chave: injeção de falhas, teste de software, validação experimental

Abstract. *Dependable systems represent a challenge for the quality assurance process. The error detection and recovery strategies build into the application must be carefully tested. Common network faults, that affect the messages exchanged between the nodes or partition the network, can be emulated in a testbed environment in which the application is run. Observing the application behavior under faults, the developer can refine the fault tolerance strategies and anticipate its real failure process.* FIONA *is a software tool aimed at testing UDP-based Java network applications. The tool injects communication faults while instrumenting the application during runtime, but without altering its source code.* FIONA *can be applied in a single network node or can operate in a distributed manner, allowing the test of large scale distributed applications.*

Keywords: fault injection, software test, experimental validation

*Financiado pelos projetos ACERTE/CNPq (472084/2003-8) e DepGriFE/HP Brasil P&D
†Bolsista do Conselho Nacional de Desenvolvimento Científico e Tecnológico

1. Introdução

Uma meta importante no desenvolvimento de um software é a sua garantia de qualidade (*quality assurance*), assegurando que o sistema se comporta de acordo com a sua especificação. Uma fase essencial para assegurar esta qualidade é através da condução de testes, onde são verificados os erros e é dada uma medida da qualidade. Um bom plano de garantia de qualidade assegura que o projeto é apropriado, a implementação é cuidadosa e o produto vai ao encontro aos requisitos estabelecidos. Um melhor plano inclui a análise de defeitos e uma melhora contínua no produto [Feldman, 2005].

Sistemas que exigem alta disponibilidade necessitam se manter em operação mesmo na ocorrência de falhas. Para isso são desenvolvidos mecanismos de tolerância a falhas, que detectam as falhas e corrigem erros ou levam o sistema a um estado seguro de parada. Nestes casos, a meta de garantia de qualidade é ainda mais crítica, pois se os mecanismos desenvolvidos não operarem corretamente, os resultados podem ser desastrosos [Carreira e Silva, 1998]. Mecanismos de tolerância a falhas devem ser exaustivamente testados antes da aplicação ser colocada em um ambiente de produção. Mesmo usando componentes de boa qualidade e boas práticas de programação e arquitetura de sistemas, assumir que falhas não ocorrem não é realista.

Injeção de falhas é uma técnica eficiente de validação experimental de software. No ambiente do sistema sob teste é emulada a ocorrência de falhas de hardware, ao mesmo tempo que o comportamento da aplicação é monitorado. A injeção de falhas tem como objetivos auxiliar na fase de teste de depuração do software, 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]. É fácil encontrar injetores de falhas desenvolvidos especificamente para uma aplicação sob teste; no entanto, esta prática aumenta o custo de programação na fase de validação se comparado a injetores mais genéricos.

Aplicações distribuídas se baseiam em comunicação através da rede e falhas nesse meio podem provocar inconsistências ou comportamento imprevisível da aplicação. Neste caso, o desafio é não apenas construir mas também testar mecanismos de tolerância a falhas que considerem a possibilidade de ocorrência de problemas de comunicação. Uma técnica para emulação deste tipo de falha é através da interceptação das mensagens trocadas pela aplicação no subsistema de troca de mensagens. A mensagem enviada ou recebida é analisada e é decidido se uma falha deve ser injetada. Exemplos de falhas são descarte e atraso de pacotes.

Este trabalho apresenta o injetor de falhas de comunicação FIONA (*Fault Injector Oriented to Network Applications*), voltado à validação de aplicações distribuídas desenvolvidas usando a plataforma Java. O foco de FIONA é o teste e a validação dos mecanismos implementados pela aplicação, e não os implementados pelo sistema operacional ou pelos protocolos por ele utilizados. A ferramenta injeta falhas nas classes de comunicação da própria máquina virtual Java (JVM), acima do nível da pilha de protocolos de comunicação do sistema operacional, tornando-a disponível para qualquer plataforma que suporte Java.

Para que um experimento envolvendo um grande número de máquinas seja conduzido de forma eficiente, FIONA oferece uma arquitetura distribuída para gerenciamento

da injeção de falhas e do monitoramento entre os nodos participantes. A ferramenta tem como característica a escalabilidade, de modo a ser uma ferramenta adaptável a ambientes de larga escala.

A Seção 2 mostra características e vantagens do método de injeção de falhas para a validação de sistemas. A Seção 3 mostra algumas ferramentas usadas para teste. Em seguida, são descritas abordagens usadas para a injeção de falhas de comunicação em aplicações Java. A Seção 5 descreve a arquitetura da ferramenta FIONA e mostra um exemplo de uso, no qual uma falha de particionamento de rede é emulada em uma aplicação de comunicação em grupo. Considerações finais são feitas na Seção 6.

2. Injeção de falhas

Uma das primeiras etapas de um mecanismo de tolerância a falhas é a *detecção* dos erros causados por falhas. Essas falhas são aleatórias devido a fadiga dos componentes, imperfeição de manufatura e eventos externos, como aquecimento, radiação eletromagnética. Essa capacidade de detecção deve ser testada antes que o sistema entre em produção para que se conheça a cobertura (*coverage*) do mecanismo e seu grau de confiança possa ser inferido. Injeção de falhas é um método muito usado por ser eficiente e simples. Os resultados alcançados complementam demais métodos, como simulações ou verificações formais.

A injeção de falhas, que interessa a este trabalho, pode ser realizada por *hardware* ou por *software*. A injeção por hardware usa equipamentos adicionais, como ponteiras ativas, para introduzir falhas diretamente no hardware do sistema alvo. Apesar de apresentar uma baixa perturbação, seu custo e seu risco de danos aos componentes eletrônicos é alto, além de apresentar uma grande latência para manifestação de erros [Hsueh et al., 1997]. A injeção por software usa código inserido na aplicação de modo a emular a falha. Mesmo podendo resultar em uma maior perturbação no comportamento da aplicação, a injeção de falhas de hardware por software se mostra mais flexível, além de ter um custo muito menor.

O código do injetor de falhas por software pode ser introduzido em tempo de compilação, onde a aplicação é modificada e o efeito da falha é fixo e programado, ou em tempo de execução, onde a falha é ativada sem modificação do código-fonte do programa alvo. A injeção em tempo de execução permite que o cenário de falhas seja alterado de acordo com o *workload*. Para implementá-la, podem-se monitorar eventos de interesse que ocorram na aplicação (como, por exemplo, exceções) ou inserir código que ative a falha antes ou depois de certas instruções – ao contrário da injeção em tempo de compilação, as instruções originais não são modificadas.

Aplicações distribuídas e de rede se caracterizam pela existência de máquinas individuais, mas conectadas. Para apresentarem alta disponibilidade, essas aplicações precisam reagir a falhas que afetem a troca de mensagem entre as várias máquinas do sistema. Para observar essa reação aplica-se injeção de falhas de comunicação, simulando problemas no envio e recebimento de mensagens pela rede. A ocorrência de falhas em sistemas distribuídos, o qual é o foco deste trabalho, já foi modelada por vários autores, entre eles Birman [1996]. Este modelo descreve as seguintes falhas: colapso, omissão de envio e recepção de mensagens, temporização, bizantinas e particionamento de rede.

Esta última é a forma mais grave de falha na rede, onde a rede se divide em subredes que estão desconectadas entre si. Máquinas dentro de uma mesma subrede conseguem se comunicar, porém todas as mensagens destinadas a máquinas fora desta subrede são perdidas.

3. Ferramentas de Injeção de Falhas

Exemplos de injetores de falhas de comunicação são ORCHESTRA [Dawson et al., 1997] e NFTAPE [Stott et al., 2000]. ORCHESTRA é uma ferramenta para teste de implementação de protocolos de comunicação. É centrada no conceito de uma camada de monitoramento e injeção de falhas chamada PFI (*Probing/Fault Injection Layer*), inserida abaixo da camada de rede sob testes e tomando ações sobre cada mensagem que por ela passa. NFTAPE usa o conceito de LWFI (*Lightweight Fault Injector*), um componente de software compacto e facilmente substituível, responsável pela injeção da falha mas não do controle e do monitoramento do experimento. Estas tarefas ficam a cargo de outros componentes do injetor e que não são alteradas entre experimentos.

Injetores como Loki [Chandra et al., 2004] adotam uma abordagem diferente, apresentando uma arquitetura distribuída, com o objetivo de testar aplicações em cenários de falhas de múltiplos nodos. Loki compartilha entre os nodos um estado global do experimento e gatilhos definidos nesse estado ativam as falhas nos processos. Apesar de ser uma abordagem flexível, sob certas situações a configuração do experimento pode tornar-se muito complexa.

FIONA é uma ferramenta de injeção de falhas de comunicação para validação de aplicações Java de larga escala. Para alcançar esse objetivo, emprega uma arquitetura distribuída para controle e monitoramento do experimento projetada para ser escalável, diferentemente de ORCHESTRA e NFTAPE. FIONA usa como abordagem para injeção de falhas a instrumentação das classes de comunicação através de uma interface nativa de depuração da plataforma Java. Seu uso é flexível, sendo possível emular um grande número de falhas em múltiplos nodos, o que possibilita a montagem de cenários de falhas complexos. Ao contrário de Loki, não é necessário especificar um estado global do sistema que sirva de gatilho para a emulação de falhas, simplificando o teste de aplicações com grande número de máquinas participantes. Além disso, FIONA não depende de nenhum recurso especial do sistema operacional ou de hardware.

4. Abordagens para injeção de falhas em Java

Existem diferentes abordagens para injeção de falhas de comunicação em aplicações Java, das quais podemos destacar reflexão computacional, programação orientada a aspectos, injeção nas camadas de mais baixo nível e injeção através de interfaces de depuração.

Ferramentas de reflexão computacional criam um novo nível de classes: além do nível base, no qual se encontram os objetos comuns, o meta-nível passa a existir. No meta-nível são criados meta-objetos, associados aos objetos base, que podem modificar seu comportamento. O injetor Jaca [Martins et al., 2002] usa a ferramenta Javassist para

criar meta-objetos associado às classes sob teste. Na extensão para falhas de comunicação [Jacques-Silva et al., 2004b], um meta-objeto intercepta os métodos de envio e recebimento de mensagens das classes de sistema responsáveis pela comunicação, injetando falhas nesses pontos. O modelo de segurança de Java, no entanto, impede que classes de sistema sejam instrumentadas. Para que isto seja feito, realiza-se um pré-processamento da aplicação, alterando-a de modo a contornar essa restrição, o que aumenta a instrusividade espacial dessa extensão do injetor e dificulta o seu uso.

Programação orientada a aspectos (*Aspect Oriented Programming*, AOP) [Kickzales et al., 1997] tem sido estudada como uma nova abordagem para injeção de falhas. Esse paradigma permite a interceptação e a alteração de métodos, em tempo de execução, sem modificar o código da aplicação sob teste. Ao mapear AOP para conceitos de injeção de falhas, determina-se que o injetor é um *crosscutting concern* da aplicação e que cada tipo de falha é um *aspecto*. FICTA [Silveira e Weber, 2005], um injetor de falhas baseado em aspectos, está sendo implementado em Java usando AspectWerkz. Mesmo já tendo sido mostrado que essa abordagem é possível, ainda é cedo para determinar suas vantagens sobre as demais.

A injeção de falhas em camadas de mais baixo nível que a JVM também pode ser usada para o teste de aplicações Java. Um exemplo de ferramenta que pode ser usada é ComFIRM [Drebes et al., 2005]. ComFIRM atua na pilha de protocolos de comunicação do sistema operacional. Como age nos níveis mais baixos da pilha de protocolos, as regras de injeção de falhas atuam em todas as mensagens trafegadas no sistema, não só nas mensagens da aplicação sob teste. Isto requer maior cuidado na configuração do injetor para que o resto do sistema não seja perturbado.

A abordagem usada por FIONA para injetar falhas é o uso de interfaces de depuração e monitoramento oferecidas pelo ambiente de execução. FIONA usa a *Java Virtual Machine Tool Interface* (JVMTI), oferecida a partir da versão 5.0 da plataforma Java. JVMTI [Sun Microsystems, 2004] provê meios para inspecionar o estado e controlar a execução de aplicações Java, permitindo seu perfilamento, depuração e monitoramento, bem como o desenvolvimento de ferramentas de controle. Classes podem ser instrumentadas no momento de sua carga pela máquina virtual, quando seus arquivos foram apenas lidos do disco, ou após já estarem carregadas e em uso. FIONA substitui as imagens das classes originais por versões instrumentadas durante sua carga.

Um cliente da interface JVMTI é denominado *agente* e é notificado pela máquina virtual Java de eventos previamente registrados. Esse agente pode, então, inspecionar e controlar a aplicação através de funções oferecidas pela JVMTI e pela JNI (*Java Native Interface*). O agente é escrito em linguagem nativa, compilado e disponibilizado em uma biblioteca que é carregada pela máquina virtual na sua inicialização.

A ferramenta FIONA é composta de um agente JVMTI, de classes de comunicação instrumentadas e de classes auxiliares para a injeção de falhas. O agente JVMTI é notificado pela JVM do carregamento das classes de sistema responsáveis pela comunicação. No lugar destas, são carregadas classes instrumentadas para injeção de falhas. O agente é codificado na linguagem C; as demais classes, em Java.

5. FIONA – *Fault Injector Oriented to Network Applications*

O uso direto dos injetores de falhas existentes, como os citados na seção 3, acarretam algumas dificuldades na condução de experimentos de testes em aplicações que executam em ambientes largamente distribuídos. A dificuldade inicia na especificação do cenário de falhas, onde cada injetor deve ser configurado individualmente. É necessário ser configurado um cenário específico para cada nodo que sofrerá injeção de falhas durante sua execução. O desafio aumenta se consideramos a configuração de uma falha que afeta mais de um nodo, como o particionamento de rede. Neste caso, a falha deve ser configurada em todos os nodos participantes. Outro empecilho diz respeito à obtenção de medidas de dependabilidade, pois considerando injetores individuais não há um log único do experimento, mas sim vários logs individuais de cada instância do injetor, potencialmente com horários diferentes.

FIONA – Fault Injector Oriented to Network Applications – é uma ferramenta baseada na interface nativa JVMTI desenvolvida para a condução de experimentos de injeção de falhas de comunicação em aplicações Java distribuídas. FIONA pode ser usada para a verificação do funcionamento correto de mecanismos de tolerância a falhas em aplicações Java distribuídas de maneira fácil e flexível.

Para emulação de um cenário de falhas em múltiplos nodos, as ferramentas de injeção de falhas de comunicação geralmente necessitam que uma instância do injetor de falhas seja ativada para cada nodo participante da aplicação. Neste caso o controle de cada injetor é feito individualmente, assim como a coleta de suas informações de monitoramento. Além disso tornar-se impraticável com o aumento no número de participantes do sistema, alguns cenários de falha, como particionamento de rede, são de difícil implementação sem coordenação entre os injetores de cada nodo. FIONA é baseada nas seguintes premissas:

- ser escalável para sistemas distribuídos de larga-escala;
- considerar um modelo de falhas consistente com este tipo de sistema;
- não alterar o código da aplicação sob teste;
- permitir a configuração de experimentos e a análise *post-mortem* de *log*s de forma centralizada.

As seções seguintes apresentam o modelo de falhas considerado, a arquitetura local e distribuída da ferramenta, e como podem ser construídos os cenários de falhas para a condução de experimentos de injeção de falhas com a ferramenta.

5.1. Modelo de falhas

Segundo Schneider [1993], um modelo de falhas é uma coleção de atributos e um conjunto de regras que governam a interação entre componentes que falharam. Um componente que apresenta falhas tem um comportamento consistente com o modelo de falhas assumido. Como o objetivo da ferramenta é o teste e validação de sistemas distribuídos, deve-se escolher um modelo que represente as falhas que comumente ocorrem nesse ambiente.

O modelo de falhas para sistemas distribuídos assumido em FIONA é baseado no definido por Birman [1996], que descreve falhas de colapso (*halt*), parada segura (*fail-stop*), omissão na transmissão e na recepção, rede, particionamento de rede, temporização e bizantinas.

Como o foco da ferramenta é o teste de sistemas distribuídos de larga escala, foram consideradas as falhas mais comuns nesse ambiente. FIONA emula falhas de colapso, de omissão, de temporização, de duplicação e de particionamento de redes.

UDP (*User Datagram Protocol*) e TCP (*Transmission Control Protocol*), são os dois protocolos de transporte mais usados sobre IP (*Internet Protocol*). FIONA injeta falhas em aplicações que usam o protocolo UDP. Este protocolo é comumente usado como base para implementação de protocolos de nível mais alto, implementados em aplicações com características de tolerância a falhas. Exemplos são o *middleware* de comunicação em grupo JGroups [Ban, 1998], que por padrão usa UDP como base para sua pilha de protocolos, e JRMS (*Java Reliable Multicast Service*) [Hanna et al., 1998], que implementa *multicast* confiável sobre UDP. O protocolo TCP apesar de ser bastante usado em aplicações ponto a ponto, por ser orientado a fluxo (stream) não é adequado para construção de sistemas de comunicação de grupo. O TCP também impõe uma semântica FIFO (First In First Out), que frequentemente não é necessária para determinadas aplicações [Wiesmann et al., 2003].

5.2. Arquitetura local

FIONA realiza a injeção de falhas através da instrumentação protocolos, os quais são implementados em classes do sistema. Falhas em comunicação usando o protocolo UDP são ativadas através da instrumentação da classe `java.net.DatagramSocket`, que contém os métodos `send()` e `receive()` para envio e recebimento de datagramas.

O injetor é composto de três partes: o agente JVMTI, o protocolo instrumentado e as classes auxiliares de injeção de falhas. O agente JVMTI realiza a instrumentação das classes de sistema e mantém a comunicação com os demais injetores da rede; o protocolo instrumentado realiza a injeção das falhas; as classes auxiliares armazenam a configuração do experimento e os *logs* de monitoramento. A Figura 1 ilustra a arquitetura local do injetor.

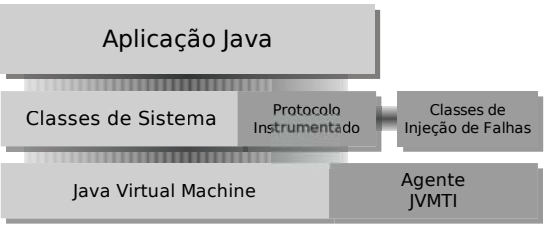

Figura 1: Arquitetura local do injetor FIONA

Na fase de inicialização da aplicação, o agente JVMTI é notificado da carga da classe `java.net.DatagramSocket` através do evento *Class File Load Hook*, capturado na fase de inicialização da máquina virtual Java. No momento da notificação, o arquivo de classe original já foi carregado em memória, mas ainda não foi processado pela máquina virtual. O agente, então, substitui essa imagem pela versão instrumentada, localizada junto aos demais arquivos do injetor. É essa versão instrumentada que será interpretada e executada, de maneira transparente à aplicação. A aplicação não necessita

```
package java.net;
import fiona.*;

public class DatagramSocket
{
   // Métodos e atributos originais da classe DatagramSocket

   public void sendOriginal(DatagramPacket p)
                            throws java.io.IOException;

   public void send(DatagramPacket p)
                            throws java.io.IOException
   {
      boolean enviar = true;
      Vector falhas = BancoFalhas.retornaFalhas(p);

      for (int i = 0; i < falhas.size(); i++) {
         Falha f = (Falha) falhas.get(i);
         if (!f.isTransmissao())
            continue;
         if ( f instanceof FalhaUdpOmissao ||
              f instanceof FalhaUdpColapso ) {
            if (f.injetar())
               enviar = false;
         }
      }
      if (enviar) {
         sendOriginal(p);
         return;
      } else {
         Monitor.log(p);
      }
   }
}
```

Figura 2: Exemplo da instrumentação do método send() para falhas de omissão e colapso

de nenhuma modificação em seu código-fonte ou *bytecode* para execução com injeção de falhas.

Na inicialização são carregadas também as classes `BancoFalhas` e `Monitor`. A primeira armazena a configuração do injetor, interage com o agente JVMTI e provê funções para controle da injeção de falhas. A segunda é responsável pelo monitoramento das falhas injetadas, gerando um *log* da execução.

Os métodos de troca de mensagens `send()` e `receive()`, instrumentados, chamam o método `BancoFalhas.retornaFalhas()`, passando como argumento referências ao pacote a ser enviado e ao socket usado. Este método retorna um vetor com as falhas que podem ser injetadas no momento, dado os endereços e portas locais e remotas em uso. Para cada falha recuperada por `retornaFalhas()`, chama-se o método `injeta()`, que retorna um booleano indicando se a falha deve ser injetada. A Figura 2 mostra um exemplo da instrumentação realizada no código do método `send()`.

Para a falha de omissão de mensagens, é configurado se sua ativação se dará no

envio ou no recebimento de uma mensagem. Este parâmetro de configuração foi introduzido para permitir o teste de comunicações que usam *multicast*. Em um multicast, se a falha for injetada no envio, nenhum participante do grupo a recebe; se for injetada no recebimento, pode-se controlar quais participantes não a recebem.

FIONA pode ser empregada para o teste de qualquer aplicação que use subclasses de `java.net.DatagramSocket`, desde que chamem os métodos `send()` e `receive()` da superclasse. O principal exemplo é a classe `java.net.MulticastSocket`, usada para realizar multicast UDP. A injeção de falhas ocorre transparentemente à aplicação, usando os mesmos mecanismos descritos acima.

A versão atual de FIONA é baseada em uma versão anterior, onde era possível apenas a injeção de falhas local Jacques-Silva et al. [2004a]. Esta versão é referenciada apenas como a arquitetura local da ferramenta, e é aqui descrita para facilitar a compreensão do funcionamento do injetor desenvolvido.

5.3. Arquitetura distribuída

Para cada processo que participa do sistema distribuído sob testes, uma instância do injetor é criada, cada uma com um estado interno individual. O objetivo de FIONA é validar sistemas de larga escala, com um grande número de participantes. Além disso, em cenários como particionamento de rede, é necessário ativar falhas em todas as máquinas participantes de forma coordenada. Nesse contexto, não é prático controlar individualmente cada injetor.

FIONA oferece uma arquitetura para controle do experimento de injeção de falhas e seu monitoramento em aplicações distribuídas. A configuração do experimento é distribuída através da rede e a ativação e desativação de falhas de particionamento de rede são iniciadas por um injetor coordenador do experimento. Da mesma forma, informações de monitoramento da aplicação são reunidos e agregados em um *log* único. A distribuição da configuração pela rede não cria um estado único global: cada máquina armazena apenas as configurações de falhas que dizem respeito a si (falhas que podem ser injetadas em pacotes enviados ou recebidos por ela). Essa filtragem das configurações diminui a intrusividade temporal, pois limita o número de regras que têm de ser analisadas para cada pacote usado pela aplicação.

Para que o injetor seja usado em aplicações com muitos nodos participantes, sua arquitetura deve ser escalável. Injeção de falhas pode ser vista como um tipo de monitoramento ativo onde, para cada evento de interesse capturado, uma ação é tomada. Portanto, a arquitetura de FIONA é inspirada em protocolos de monitoramento. Dois protocolos usados em sistemas distribuídos de larga escala foram considerados: GRM e Ganglia.

A estratégia implementada por GRM [Balaton et al., 2001] consiste em estabelecer uma hierarquia entre os monitores, que são divididos em monitores *locais*, monitores de *site* e monitor *principal*. Os monitores locais são colocados em cada máquina do sistema distribuído e passam suas informações para o monitor de *site*. Cada monitor de *site* é responsável por coletar as informações do conjunto de computadores a ele associados (por exemplo, de um centro de computação) e as repassa para o monitor principal, esse responsável por coordená-los e agregar as informações globais. Ganglia [Massie et al.,

2003] realiza o monitoramento de *clusters* de computadores. Com essa ferramenta, é possível monitorar mais de um *cluster* ao mesmo tempo e centralizar em uma única interface suas informações. Ganglia usa conexões TCP para comunicação entre *clusters* diferentes, e para monitoramento dos nodos individuais, multicast UDP. O uso de protocolos diferentes, dependendo da confiabilidade da rede em uso, reduz o custo das operações de comunicação do injetor de falhas.

FIONA combina a hierarquia de GRM com a estratégia de protocolos de Ganglia. A Figura 3 mostra a adaptação da arquitetura de monitoramento para uma arquitetura de injeção distribuída de falhas. A comunicação entre os injetores locais e os injetores de *site* é feita por multicast UDP, e a comunicação entre os injetores de *site* e o injetor principal é feita por conexão TCP. Ela é realizada por uma thread criada pelo agente JVMTI, fora do ambiente Java, para que não seja interferida pela injeção de falhas causada pela ferramenta.

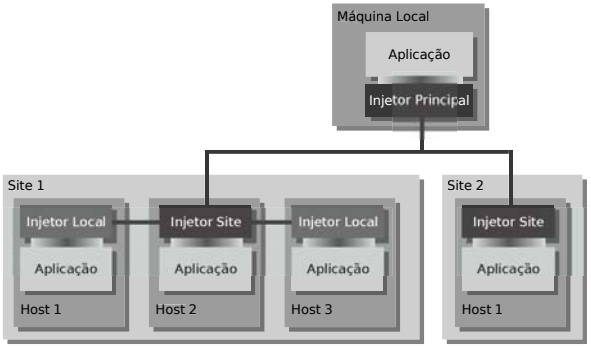

Figura 3: Arquitetura distribuída do injetor FIONA

FIONA mantém um *log* com todas as falhas injetadas ao longo da execução em cada nodo participante: cada chamada aos métodos `send()` ou `receive()` que teve uma falha injetada é registrada. Esse *log* permite a análise *post-mortem* do comportamento da aplicação. No final da execução, cada injetor envia o seu *log* para o injetor imediatamente superior na hierarquia. Se o nodo é um injetor principal ou de *site*, no final da execução da aplicação, o injetor espera até que todos os nodos conectados a ele enviem seus *logs*. Os *logs* de aplicação são dependentes desta e não são tratados pelo sistema de monitoramento de FIONA; devem ser passados para uma ferramenta de análise de dependabilidade à parte, juntamente com os logs da injeção de falhas.

Cada evento é registrado com o tempo local do nodo. Para que se possa gerar uma linha de tempo global de todo o experimento, usamos o algoritmo proposto por Maillet e Tron [Maillet e Tron, 1995], baseado na diferença de relógio entre uma máquina de referência e cada máquina de um sistema distribuído. Essa diferença é medida no início e no final do experimento. As diferenças individuais são armazenadas na máquina de referência e usadas posteriormente para a correção dos tempos registrados nos *logs*.

Este método de sincronização é implementado por um programa auxiliar. Este programa é executado duas vezes, antes do início do experimento e logo após sua fina-

lização. Nestas fases, uma operação de *ping-pong* é feita, entre a máquina de referência e os outros nós do sistema, para recolher dados relacionados ao tempo de cada máquina do sistema distribuído. Esses dados são usados para gerar os parâmetros de correção de tempo de cada máquina. Após estas operações, o programa lê os *logs* coletados por FIONA e ajusta seus horários baseado nos parâmetros de correção. A saída é um *log* único ordenado por tempo com todas as atividades de injeção de falhas. Esse algoritmo é eficiente para sistemas de pequena e média escala; no entanto, como não necessitamos de uma resolução fina de tempo, entendemos que é aceitável para ser aplicados também a sistema de larga escala.

A simulação de um particionamento de rede é de fácil realização com essa arquitetura distribuída. Em um arquivo de configuração, são descritas as partições que devem ser formadas, indicando os endereços das máquinas na forma de subredes IP ou seus nomes. No início do experimento, a configuração é espalhada para as máquinas locais. A falha é ativada por tempo pelo injetor principal, que envia mensagens de controle para os injetores inferiores da hierarquia.

5.4. Configuração de cenários 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. O cenário, que servirá como carga de falhas para a aplicação a ser testada com FIONA, é codificado em um arquivo de configuração, processado pelo injetor principal no momento de sua inicialização. Em cada linha do arquivo é declarada uma falha e seus parâmetros específicos.

A arquitetura distribuída de FIONA é usada para transmitir a configuração entre as máquinas participantes. O injetor principal a envia para os injetores de *site*, que a repassam para os injetores locais. É possível criar um arquivo de configuração de falhas específico para os injetores de *site* e locais; no entanto, devido à arquitetura do sistema, isso não é necessário. Apenas as falhas que dizem respeito às máquinas destino são enviadas.

Para auxiliar o usuário na especificação de um cenário de falhas, a ferramenta FIONA provê uma interface gráfica para edição dos arquivos de configuração. Uma captura de tela pode ser vista na Figura 4, que mostra a interface para a configuração de uma falha de particionamento de rede. Neste caso, o usuário especifica as máquinas que fazem parte de cada uma das partições através de seu nome ou endereço IP e as condições de início da falha (tempo inicial e final em segundos a partir do início da execução do injetor). Diferentes falhas têm diferentes opções de configuração: por exemplo, para a falha de omissão de mensagens são especificados endereços e portas de origem e destino, percentual das mensagens que serão perdidas e condições de início e término de falha.

A configuração mostrada na Figura 4 especifica um cenário de falha de particionamento de rede com duas partições, cada uma com duas máquinas. A primeira partição contém as máquinas `buick` e `bentley`; a segunda, `maverick` e `corvette`. A linha de configuração resultante é a seguinte:

```
UdpNetworkPartitioningFault:50:15000:
 buick,bentley:maverick,corvette:
```

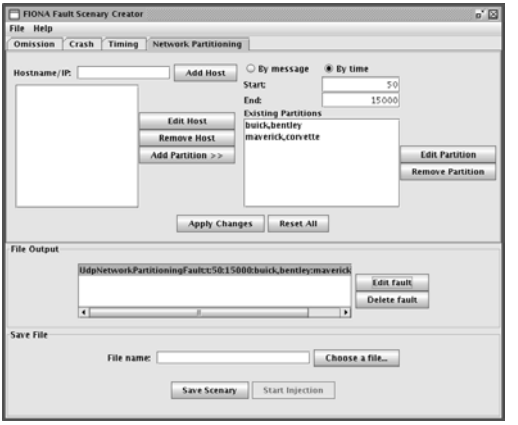

Figura 4: Interface gráfica de FIONA

A falha foi configurada para iniciar 50 segundos após o início do experimento. Nesses 50 segundos iniciais, todas as máquinas comunicam-se entre si. Após a ativação da falha, a rede se divide nas duas partições citadas acima.

Para carregar FIONA ao executar uma aplicação, basta solicitar à máquina virtual o carregamento do agente JVMTI. Isto é realizado através da passagem do parâmetro `-agentlib` para a máquina virtual Java, especificando a localização do agente e opções como tipo do injetor (principal, *site* ou local) e nome do arquivo de falhas. A interface gráfica pode ser usada para iniciar a aplicação com o injetor: os parâmetros são construídos e passados para a máquina virtual automaticamente. A interface, se usada, é iniciada apenas na máquina coordenadora do experimento, não sendo necessária nos demais nodos. É interessante notar que a única diferença entre um experimento de injeção de falhas e o disparo normal da aplicação, sem o injetor, é apenas a presença do parâmetro que inicia o agente JVMTI. Isto facilita o uso da ferramenta, permitindo comparar de maneira simples o comportamento da aplicação com e sem falhas.

Para demonstração da ferramenta, a falha descrita acima foi injetada em uma aplicação de teste do *middleware* de comunicação em grupo JGroups que implementa um programa de desenho compartilhado. A Figura 5 mostra uma execução do exemplo descrito acima. Cada janela é uma instância do programa, executando em uma máquina diferente. Durante os 50 segundos iniciais, quando todas as máquinas comunicavam-se entre si, foi desenhada a primeira coluna de letras. Cada letra foi desenhada em uma instância diferente e foi transmitida para as demais. Após 50 segundos, a falha foi ativada e a rede dividida nas partições especificadas. Cada letra da segunda coluna foi também desenhada em uma instância diferente, mas os desenhos somente eram transmitidos dentro de cada partição da rede.

O exemplo mostra que a emulação de uma falha de particionamento de rede pode ser realizada facilmente usando a ferramenta FIONA. Esse exemplo foi escolhido por ser simples visualizar uma situação de defeito: o observador pode perceber a forma como a

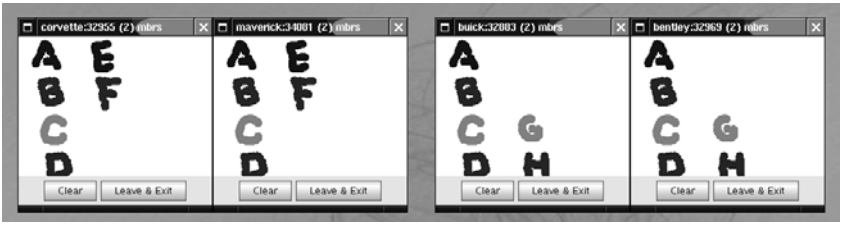

Figura 5: Exemplo de um cenário de particionamento de rede

aplicação distribuída se comporta em um ambiente sujeito a essa falha. Exemplos de outras falhas emuladas por FIONA podem ser encontrados em Jacques-Silva et al. [2004a].

6. Conclusões

A demanda de determinados sistemas computacionais por alta-disponibilidade é grande. A garantia de qualidade nesses sistemas é essencial e técnicas de teste de software, como injeção de falhas, são de extrema importância.

Injeção de falhas é uma técnica eficiente e muito usada para a validação de mecanismos de tolerância a falhas de aplicações. Com ela, é possível analisar o comportamento de uma aplicação quando falhas ocorrem, sem a necessidade de esperar pela ocorrência aleatória de falhas reais. Este trabalho apresenta FIONA, um injetor de falhas de comunicação para aplicações distribuídas Java de pequena e larga escala.

A ferramenta FIONA pode ser usada tanto para auxiliar no desenvolvimento de software como para obtenção de medidas de dependabilidade. Ferramentas deste tipo são importantes na fase de teste do software, na depuração dos mecanismos de tolerância a falhas desenvolvidos. O desenvolvedor pode testar sua aplicação com o injetor até o seu comportamento estar de acordo com a sua especificação.

O uso de ferramentas de injeção de falhas são essenciais para a obtenção de medidas de dependabilidade. A obtenção de métricas é importante para duas situações: assegurar que o software desenvolvido obedece a determinadas exigências de funcionamento, mesmo em condição de falhas; e permitir a avaliação de componentes desenvolvidas externamente, como componentes de software ou middleware, verificando se estas são adequadas para um determinado projeto, ou até mesmo comparando componentes similares. FIONA permite a avaliação de componentes externos pois não exige que as aplicações sob teste tenham seu código-fonte disponível.

FIONA oferece uma arquitetura distribuída para configuração, controle e monitoramento do experimento de experimentos de injeção de falhas que envolva de dois a vários nodos, podendo ser usada para teste de aplicações distribuídas de larga escala. Ela foi inspirada em ferramentas de monitoramento de *clusters* e *grids* computacionais. FIONA realiza monitoramento através de *logs* que permitem a análise *post-mortem* da aplicação.

A implementação do injetor é realizada com o uso de JVMTI, uma interface nativa de depuração e monitoramento da plataforma Java que permite a instrumentação de classes e controle de execução das aplicações. FIONA é composto de um agente JVMTI,

responsável pelo controle e comunicação, e classes Java para instrumentação e configuração.

A ferramenta é independente de sistema operacional e arquitetura, de simples operação e flexível. FIONA é codificada principalmente em Java, usando pouco código nativo. A configuração de um experimento pode ser realizada através de uma interface gráfica e a inicialização do injetor de falhas não requer alterações na aplicação e a ela é transparente.

Referências

Balaton, Z., Kacsuk, P., Podhorszki, N., e Vajda, F. (2001). From cluster monitoring to grid monitoring based on GRM. In *Proceedings of Euro-Par 2001*, Manchester, UK.

Ban, B. (1998). JavaGroups - group communication patterns in Java. Technical report, Department of Computer Science, Cornell University.

Birman, K. P. (1996). *Building Secure and Reliable Network Applications*. Manning Publications, Co, Greenwich.

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

Chandra, R., Lefever, R. M., Joshi, K. R., Cukier, M., e Sanders, W. H. (2004). A global-state-triggered fault injector for distributed system evaluation. *IEEE Transactions on Parallel and Distributed Systems*, 15(7):593–605.

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

Drebes, R. J., Leite, F. O., Jacques-Silva, G., Mobus, F., e Weber, T. S. (2005). ComFIRM: a communication fault injector for protocol testing and validation. In *Digest of Papers, The 6th IEEE LATW*, páginas 115–120, Salvador, Bahia, Brazil.

Feldman, S. (2005). Quality assurance: Much more than testing. *Queue*, 3(1):26–29.

Hanna, S., Kadansky, M., e Rosenzweig, P. (1998). The Java Reliable Multicast Service: A reliable multicast library. Technical report, Sun Microsystems. Disponível em: <`http://www.experimentalstuff.com/sunr/techrep/1998/abstract-68.html`>.

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

Jacques-Silva, G., Drebes, R. J., Gerchman, J., e Weber, T. S. (2004a). FIONA: A fault injector for dependability evaluation of Java-based network applications. In *Proc. of the 3rd IEEE Intl. Symposium on Network Computing and Applications*, páginas 303–308, Cambridge, MA. IEEE Computer Society Press.

Jacques-Silva, G., Moraes, R. L. O., Weber, T. S., e Martins, E. (2004b). Validando sistemas distribuídos desenvolvidos em Java utilizando injeção de falhas de comunicação

por software. In *Anais do V Workshop de Testes e Tolerância a Falhas*, páginas 53–64, Gramado, Brasil.

Kickzales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., e Irwin, J. (1997). Aspect-oriented programming. In *Proceedings of the ECOOP'97*, Jyväskylä, Finland.

Maillet, E. e Tron, C. (1995). On efficiently implementing global time for performance evaluation on multiprocessor systems. *Journal of Parallel and Distributed Computing*, 28(1):84–93.

Martins, E., Rubira, C. M. F., e Leme, N. G. M. (2002). Jaca: A reflective fault injection tool based on patterns. In *Proceedings of DSN 2002*, páginas 483–487, Washington, USA.

Massie, M. L., Chun, B. N., e Culler, D. E. (2003). The Ganglia distributed monitoring system: Design, implementation, and experience. Technical report, University of California, Berkeley.

Schneider, F. B. (1993). What good are models and what models are good? In Mullender, S., editor, *Distributed Systems*, páginas 17–26. Addison-Wesley, Workingham, 2nd edition.

Silveira, K. K. e Weber, T. S. (2005). Um injetor de falhas de comunicação construído usando programação orientada a aspectos. In *Anais do V Workshop de Testes e Tolerância a Falhas*, páginas 55–66, Fortaleza, CE.

Stott, D. T., Floering, B., Burke, D., Kalbarczyk, Z., e Iyer, R. K. (2000). NFTAPE: A framework for assessing dependability in distributed systems with lightweight fault injectors. In *Proceedings of IPDS 2000*, páginas 91–100, Chicago, USA.

Sun Microsystems (2004). Java Virtual Machine Tool Interface. Disponível em: <`http://java.sun.com/j2se/1.5.0/docs/guide/jvmti/`>.

Wiesmann, M., Défago, X., e Schiper, A. (2003). Group communication based on standard interfaces. In *Proceedings of NCA'03*, Cambridge, MA.