Implementação de um Injetor de Falhas de Comunicação para Aplicações Java Baseado em JVMTI*

Júlio Gerchman[1], Gabriela Jacques-Silva[1†], Taisy Silva Weber[1]

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

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

Resumo. *O uso de aplicações distribuídas em Java em sistemas de missão crítica exige a validação de seus mecanismos de tolerância a falhas. Para realizar essa validação experimentalmente foi desenvolvido FIONA, um injetor de falhas de comunicação para aplicações Java. FIONA utiliza a nova interface de controle e depuração de máquinas virtuais JVMTI disponibilizada com a versão 1.5 da J2SE. O uso dessa interface proporciona a realização de experimentos de injeção de falhas de modo transparente à aplicação em teste, sem necessidade de alteração do código-fonte.*

1. Introdução

Aplicações distribuídas representam um desafio à área de tolerância a falhas, pois além de ser necessário considerar problemas internos a cada nó participante, erros na comunicação também podem ocorrer. Para que este tipo de aplicação seja utilizada em sistemas de alta disponibilidade, é necessária a implementação de técnicas que permitam o funcionamento correto do sistema mesmo na ocorrência falhas. Uma etapa fundamental no desenvolvimento de sistemas tolerantes a falhas é a fase de validação, na qual é verificado o comportamento do sistema em vários cenários de falha. Não verificar os mecanismos de detecção e correção de erros pode permitir a ocorrência de comportamentos inesperados e errôneos já em fase operacional do *software*.

A injeção de falhas é uma técnica de validação experimental de sistemas, na qual falhas são introduzidas controlada e artificialmente no sistema, tendo sua resposta monitorada [Hsueh *et al.* 1997]. Desta maneira, é possível testar a eficiência dos mecanismos de tolerância a falhas implementados e também determinar medidas de dependabilidade, como a cobertura da detecção de erros e o impacto no desempenho dos mecanismos desenvolvidos.

Este trabalho apresenta o injetor de falhas de comunicação FIONA (*Fault Injector Oriented to Network Applications*) destinado a validação de aplicações distribuídas desenvolvidas usando a plataforma Java, que é largamente usada no desenvolvimento deste tipo de sistema devido à sua portabilidade. Esta ferramenta injeta as falhas na própria

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

máquina virtual Java (JVM), acima do nível da pilha de protocolos de comunicação do sistema operacional, sendo assim facilmente portável.

Para permitir a injeção de falhas é usada a interface de programação nativa *Java Virtual Machine Tool Interface* (JVMTI) [Sun Microsystems 2004]. A JVMTI é uma interface que possibilita tanto monitoramento como controle de execução de aplicações em execução em uma JVM. Um dos recursos disponibilizados pela JVMTI é a instrumentação de *bytecodes*, que neste trabalho é usado para a inserção de código de injeção de falhas na aplicação.

2. Injeção de Falhas

A injeção de falhas pode ser realizada de várias formas, sendo as mais comuns por simulação, por *hardware* ou por *software*. A ferramenta apresentada neste trabalho se enquadra na última categoria. Uma ferramenta de injeção de falhas por *software* geralmente é um trecho de código que usa todos os ganchos (*hooks*) possíveis do processador e do sistema para criar um comportamento incorreto de maneira controlada [Carreira e Silva 1998]. A estratégia de implementação por *software* é mais flexível e tem menor custo que os outros métodos, além de poder simular tanto problemas de *software* como de *hardware*. Em FIONA são simuladas falhas de comunicação, sendo que estas podem tanto ser consideradas falhas de *hardware* como também resultado de alguma falha de *software*. Exemplos deste tipo de falhas incluem mensagens que chegam ao destinatário fora de ordem ou que são perdidas.

Outros exemplos de ferramentas de injeção de falhas de comunicação são ORCHESTRA [Dawson *et al.* 1996] e ComFIRM [Barcelos *et al.* 2000]. A primeira ferramenta foi desenvolvida especificamente para teste de dependabilidade de protocolos distribuídos. A injeção de falhas é através da inserção de uma camada na pilha de protocolos, chamada de PFI (*Protocol Fault Injection*). A ferramenta ComFIRM (*Communication Fault Injection through OS Resources Modification*) injeta falhas diretamente no núcleo do sistema operacional Linux, no nível mais baixo do tratamento de mensagens pelo subsistema de rede. O código da ferramenta é inserido diretamente no núcleo do sistema operacional, o que diminui consideravelmente a intrusão temporal da ferramenta no sistema sob teste. Uma ferramenta para injeção de falhas em Java é Jaca, que usa reflexão computacional para alterar o comportamento da aplicação. Jaca injeta falhas de interface e recentemente foi estendida para injeção de falhas de comunicação [Jacques-Silva *et al.* 2004]. Tal extensão exige uma fase de pré-processamento para alteração das referências das classes de comunicação para as classes instrumentadas. O pré-processamento pode ser tanto para a alteração do código da aplicação quanto para a alteração de *bytecodes*.

FIONA usa como abordagem de injeção de falhas uma interface de programação nativa padrão para desenvolvimento de ferramentas de monitoramento e depuração em Java, chamada JVMTI (Java Virtual Machine Tool Interface). FIONA baseia sua implementação em um agente JVMTI, desenvolvido em C, juntamente com classes de injeção de falhas desenvolvidas em Java. Através do uso da linguagem nativa no menor número possível de casos, FIONA mantém sua portabilidade para todas as máquinas virtuais que implementam JVMTI.

3. JVMTI – Java Virtual Machine Tool Interface

A *Java Virtual Machine Tool Interface* (JVMTI) é uma interface de programação para monitoração e controle de execução de aplicações Java, que permite a interceptação de eventos da máquina virtual, instrumentação das aplicações em tempo de execução e é desenvolvida diretamente pela Sun, a criadora da plataforma. A JVMTI proporciona uma interface com a máquina virtual Java, permitindo o seu controle, depuração, perfilamento e monitoramento, entre outras atividades. Esta interface é uma parte da *Java Platform Debugger Architecture* (JPDA), que provê interfaces de programação para a depuração de aplicações. JVMTI é oferecida a partir da versão 1.5 da API e vem substituir as interfaces *Java Virtual Machine Profiling Interface* (JVMPI) e *Java Virtual Machine Debugger Interface* (JVMDI), que serão descontinuadas nas próximas versões.

O componente cliente das funções da JVMTI é chamado *agente*. O agente é uma biblioteca dinâmica que é notificada de eventos de interesse através de *eventos* como, por exemplo, o carregamento de uma classe ou o início da execução de uma *thread*. Após a notificação, o agente pode chamar funções que podem alterar ou inspecionar em maior detalhe o estado da máquina virtual. O agente executa juntamente com o código da máquina virtual Java, proporcionando controle com uma intrusão mínima na execução da aplicação. Como é possível ainda ele ser controlado por um processo em separado, pode ser mantido compacto, interferindo ainda menos na execução [Sun Microsystems 2004].

O uso da JVMTI descarta a necessidade de alteração e recompilação das aplicações alvo para execução de experimentos de injeção de falhas, pois toda a instrumentação pode ser realizada durante a execução. O agente JVMTI é carregado opcionalmente no disparo da máquina virtual. Essas características facilitam e flexibilizam o uso de um injetor programado com essa interface.

4. FIONA – Fault Injector Oriented to Network Applications

FIONA – *Fault Injector Oriented to Network Applications* – é uma ferramenta desenvolvida para condução de experimentos de injeção de falhas em aplicações Java baseada em JVMTI com o objetivo específico de implementação de falhas de comunicação. FIONA pode ser utilizado para a verificação do funcionamento correto de mecanismos de tolerância a falhas em aplicações Java distribuídas de uma maneira fácil e flexível.

O modelo de falhas implementado trata de falhas que podem ocorrer no protocolo UDP (*User Datagram Protocol*). Esse protocolo é não confiável e não orientado a conexão; no entanto, é comumente utilizado como base para implementação de protocolos que implementam a confiabilidade necessária através de camadas superiores adicionais. Um desses exemplos é o *middleware* de comunicação de grupo JGroups [Ban 1998], que usa UDP por *default* na base de sua pilha de protocolos. Este é um caso onde um injetor de falhas na comunicação no protocolo UDP é de grande utilidade.

4.1. Arquitetura e Funcionamento

FIONA implementa injeção de falhas na comunicação pelo protocolo UDP através da instrumentação da classe `java.net.DatagramSocket`, que contém os métodos `send()` e `receive()` usados para enviar e receber datagramas desse protocolo. A

instrumentação é realizada pelo agente JVMTI durante o carregamento desta classe. Na fase de inicialização também são carregadas outras classes auxiliares, necessárias para a configuração do injetor e controle do experimento.

O agente JVMTI é notificado do carregamento da classe `java.net.DatagramSocket` através do evento *Class File Load Hook*, capturado na fase de inicialização da máquina virtual (*start phase*), antes da fase de execução da aplicação. No momento da notificação, uma imagem do arquivo de classe já foi carregado em memória, porém ainda não foi processado pela máquina virtual. O agente então substitui essa imagem por uma versão instrumentada, localizada junto aos outros arquivos do injetor. É essa versão instrumentada que será interpretada e executada de modo transparente à aplicação. Assim, ela não necessita de modificações em seu código-fonte para execução com injeção de falhas.

Quando a máquina virtual entra na fase de execução (*live phase*), a classe `BancoFalhas` é instanciada. Essa classe, ao ser instanciada, lê o arquivo de configuração do injetor e armazena quais falhas devem ser ativadas durante a execução da aplicação. `BancoFalhas` também provê métodos estáticos para acesso às falhas carregadas, que serão usados pela classe `java.net.DatagramSocket` instrumentada.

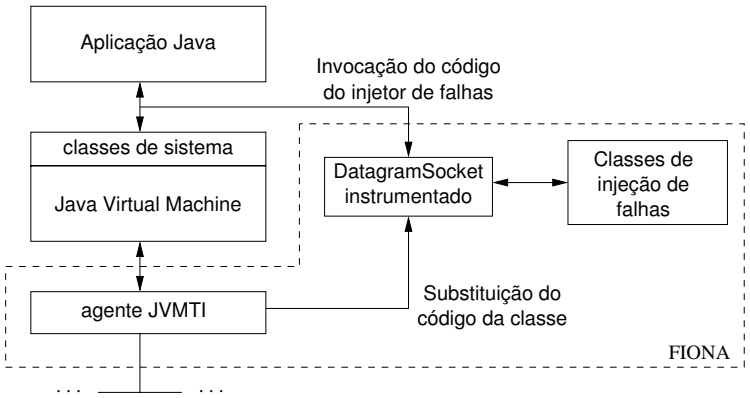

Figura 1: Arquitetura de FIONA

Como dito anteriormente, o envio e recebimento de um datagrama UDP pela aplicação se dá através dos métodos `send()` e `receive()` da classe `java.net.DatagramSocket`. A chamada a esses métodos executará o código instrumentado, carregado anteriormente pelo agente JVMTI. A classe chama o método `BancoFalhas.retornaFalhas()`, passando como argumento o pacote a ser enviado. Este método retorna um vetor com falhas (objetos de especializações da superclasse `Falha`) que têm a possibilidade de serem injetadas no momento, dado o endereço e porta do remetente e do destinatário do datagrama.

4.2. Modelo de Falhas e Configuração do Injetor

As falhas consideradas por FIONA são baseadas no modelo de falhas para sistemas distribuídos definido por Cristian [Cristian 1991], que descreve falhas de omissão, de temporização, de resposta e de colapso. As falhas de omissão ocorrem quando um servidor não responde a uma requisição; as de temporização, quando a resposta ocorre fora

do intervalo especificado, situação geralmente associada a problemas de desempenho; as de resposta, quando a mensagem de retorno é incorreta por apresentar um valor ou uma transição de estado errôneos; as de colapso, quando o servidor pára totalmente de responder. Esse modelo foi escolhido porque, além dessas falhas serem muito comuns em um ambiente de aplicações distribuídas, o protocolo UDP não as trata.

Foram implementadas em FIONA falhas de omissão no envio e recebimento de mensagens, temporização e colapso de *host*. Durante a execução dos métodos `send()` ou `receive()`, as falhas definidas na configuração do injetor são testadas para verificar se devem ser ativadas no momento através de seu método `injeta()`, que retorna um valor booleano. Esse método é implementado de forma específica para cada classe de falha.

Caso pelo menos uma das falhas de omissão deve ser injetada, ou o pacote não é enviado (no método `send()`) ou é descartado e torna-se a esperar por um novo (no método `receive()`). No envio de pacotes, caso nenhuma falha de omissão deva ser injetada mas uma de temporização sim, uma *thread* é criada para enviar o pacote após um atraso estabelecido. Essa *thread* é criada para que o fluxo de execução retorne para a aplicação logo após a chamada a `send()` e se encerra após o envio efetivo. Se nenhuma falha deve ser injetada, o pacote é enviado normalmente.

Através de FIONA também é possível validar protocolos que utilizam multicast UDP. Como a classe Java que implementa essa funcionalidade, `java.net.MulticastSocket`, é subclasse de `java.net.DatagramSocket` e os métodos `send()` e `receive()` não são reimplementados, a injeção de falhas acontece transparentemente, usando o mesmo mecanismo. Esta característica pode ser explorada para fazer teste em aplicações que usem primitivas de multicast como base para camadas que implementam a confiabilidade em níveis superiores da pilha de protocolos.

As falhas que devem ser ativadas durante a execução da aplicação são especificadas em um arquivo texto contendo sua configuração. Esse arquivo é lido e interpretado por FIONA quando a classe `BancoFalhas` é instanciada, no início da execução da aplicação. Em cada linha é especificada uma falha e seus parâmetros. A seguir vemos um exemplo de configuração de falha de temporização:

```
FalhaUdpTemporizacao:<repeticao>:<inicio>:<taxa_falhas>:
<atraso_minimo>:<atraso_maximo>:<host_origem>:<porta_origem>:
<host_destino>:<porta_destino>:
```

O primeiro campo especifica qual a falha está sendo configurada, neste caso, uma falha de temporização. O campo `repeticao` especifica o padrão de repetição da falha: transitória (`t`, apenas uma ocorrência, na mensagem cujo número é especificado no campo `inicio`), permanente (`p`, a partir da mensagem de número especificado em `inicio` a falha sempre é ativada) ou intermitente (`i`, ocorre com uma probabilidade especificada no campo `taxa_falhas`). Os campos de atraso mínimo e máximo são particulares da falha de temporização. O *socket* no qual a falha será injetada é definido pelos campos de *hosts* e portas origem e destino.

5. Conclusões e Trabalhos Futuros

O artigo apresentou a implementação do injetor de falhas de comunicação FIONA. FIONA injeta falhas de comunicação em aplicações Java que utilizam o protocolo UDP de maneira completamente transparente para a aplicação, permitindo validar os mecanismos de tolerância a falhas desta.

A implementação de FIONA se deu através do uso da nova interface de controle, monitoramento e depuração da plataforma Java, a JVMTI. Essa interface permite a criação de programas (os agentes) capazes de inspecionar e alterar o estado de uma máquina virtual sem a interferência da aplicação. FIONA foi utilizado para o teste do mecanismo de detecção e recuperação de mensagens em uma aplicação tipo *shared whiteboard*, no qual demonstrou sua eficácia e facilidade de uso.

Trabalhos futuros seguem duas linhas complementares. Uma delas é a extensão da ferramenta para incorporação de modelos de falhas para o protocolo TCP e para o mecanismo de RMI (*Remote Method Invocation*). A outra é a extensão da ferramenta de modo a realizar a injeção distribuída de falhas, visando facilitar a condução de experimentos de validação em aplicações distribuídas, como ferramentas para *grid computing* e sistemas que utilizam comunicação em grupo.

Referências

Barcelos, P. P. A.; Leite, F. O.; Weber, T. S. *Building a Fault Injector to Validate Fault Tolerant Communication Protocols*. In Proceedings of International Conference on Parallel Computing Systems. Ensenada, México. Agosto 1999.

Ban, B. *JavaGroups - Group Communication Patterns in Java*. Department of Computer Science, Cornell University. Julho 1998.

Carreira, J.; Silva, J. G. *Why do Some (weird) People Inject Faults?* ACM SIGSOFT, Software Engineering Notes, Volume 23, Número 1, pp. 42-43. Janeiro 1998.

Cristian, F. *Understanding Fault-Tolerant Distributed Systems*. Communications of the ACM, Volume 34, Número 2, pp. 56-78. Fevereiro 1991.

Dawson, S.; Jahanian, F.; Mitton, T. *ORCHESTRA: A Probing and Fault Injection Environment for Testing Protocol Implementations*. In Proceedings of IPDS'96. Urbana-Champaign, Estados Unidos. Setembro 1996.

Hsueh, M.-C.; Tsai, T.; Iyer, R. *Fault Injection Techniques and Tools*. IEEE Computer, Volume 30, Número 4, pp. 75-82. Abril 1997.

Jacques-Silva, G.; Moraes, R. L. O.; Weber, T. S.; Martins, E. *Validando Sistemas Distribuídos Desenvolvidos em Java Utilizando Injeção de Falhas de Comunicação por Software*. Anais do V Workshop de Testes e Tolerância a Falhas. Gramado, Brasil. Maio 2004.

Sun Microsystems, *Java Virtual Machine Tool Interface*. `http://java.sun.com/j2se/1.5.0/docs/guide/jvmti/`. 2004.