AFIDS - Arquitetura para Injeção de Falhas em Sistemas Distribuídos

Irineu Sotoma
Taisy Silva Weber

Curso de Pós Graduação em Ciência da Computação
Universidade Federal do Rio Grande do Sul
Av. Bento Gonçalves 9500, Caixa Postal 15064 - 91501-970 Porto Alegre - RS
irineu@main.unicruz.tche.br, taisy@inf.ufrgs.br

Resumo

Injeção de falhas implementada por software é um dos principais métodos usados para validar protocolos tolerantes a falhas em sistemas distribuídos. Contudo, não existem ferramentas genéricas para as várias plataformas de software e hardware existentes. As ferramentas são dirigidas a um hardware ou a algum sistema operacional específico. Continuamente novas ferramentas devem ser construídas. Não há também nenhuma biblioteca de classes orientada a objetos disponível para auxiliar desenvolvedores na construção da sua própria ferramenta de injeção de falhas. AFIDS (Architecture for Fault Injection in Distributed Systems) é uma arquitetura orientada a objetos que visa auxiliar a suprir essa lacuna.

AFIDS pretende fornecer uma estrutura básica que atende à geração de parâmetros de falhas, ao controle da localização, tipo e tempo da injeção de falhas, à coleta de dados do experimento e à análise dos dados coletados de modo a obter medidas de dependabilidade sobre o protocolo tolerante a falhas. Ou seja, AFIDS pretende ser um *framework* para a construção de ferramentas de injeção de falhas. A aplicação de AFIDS é ilustrada por uma ferramenta de injeção de falhas que utiliza um objeto injetor em cada um dos processos que compõem o protocolo sob teste. AFIDS e esta ferramenta são implementadas em C++ usando *sockets* no sistema operacional Linux.

Abstract

Software-implemented fault injection is one of the main methods used to validate fault tolerant protocols in distributed systems. However, there is no generic tool for the many existent hardware and software platforms. The tools are specific for a hardware or operating system. Continuously, new tools are built. No object-oriented class library is available for aid developers in the construction of their fault injection tool. AFIDS (Architecture for Fault Injection in Distributed Systems) is an object-oriented architecture that aims to fill this gap.

AFIDS intends to provide a basic structure that addresses the generation of fault parameters, the control of the location, type and time of the fault injection, the data collected, and the analysis of collected data in order to obtain dependability measures about the fault tolerant protocol under test. Then AFIDS intends to be a *framework* for fault injection tools construction. A fault injection tool, that uses one injector object in each one of the processes that composes the protocol under test, illustrates the application of AFIDS. AFIDS and this tool are implemented in C++ using *sockets* on Linux operating system.

1. Introdução

Uma forma de validar o projeto e implementação de um protocolo tolerante a falhas é através de ferramentas de injeção de falhas. Estas ferramentas injetam falhas durante a execução do protocolo e fornecem dados sobre seu comportamento na presença das falhas injetadas. A análise destes dados indica se o protocolo atende à especificação, ou seja, se realmente mascara ou recupera das falhas a que se propôs.

Injetores de falhas implementados por software emulam falhas de hardware e software e aceleram sua ocorrência sem a necessidade de hardware adicional. Muitas ferramentas dessa classe foram construídas para atender plataformas específicas. Por exemplo, EFA [ECH 92] executa sobre o sistema operacional distribuído A/ROSE. SFI [ROS 93] e DOCTOR [HAN 93] executam sobre HARTS. Algumas ferramentas precisam de um outro sistema operacional adicional para possibilitar a injeção de falhas. DOCTOR [HAN 93], por exemplo, usa o sistema operacional de rede x-kernel.

Na prática, cada nova plataforma desenvolvida exige o desenvolvimento de uma nova ferramenta de injeção de falhas, ou adaptações profundas nas existentes, com baixo reaproveitamento. Não existem injetores de falhas comerciais. Não há também disponível nenhuma biblioteca orientada a objetos para auxiliar pesquisadores a construir sua própria ferramenta de injeção de falhas. AFIDS (Architecture for Fault Injection in Distributed Systems) pretende suprir parte dessa lacuna fornecendo uma estrutura básica de suporte ao desenvolvimento de injetores de falhas. AFIDS aborda: a geração de parâmetros de falhas para o experimento; o controle da localização, tipo e tempo da injeção de falhas; a coleta de dados do experimento; a injeção efetiva da falha e finalmente a análise dos dados coletados de modo a obter medidas de dependabilidade sobre o protocolo tolerante a falhas.

AFIDS leva em consideração várias questões de projeto que foram levantadas através da análise de oito ferramentas de injeção de falhas por software para sistemas distribuídos: FIAT [SEG 88], EFA [ECH 92], SFI [ROS 93], DOCTOR [HAN 93], PFI [DAW 94], CSFI [CAR 95], SockPFI [DAW 95] e ORCHESTRA [DAW 96].

Este artigo aborda ferramentas de injeção de falhas implementadas em software aplicadas a sistemas distribuídos e apresenta uma arquitetura para a construção destas ferramentas. A aplicação de AFIDS é ilustrada por uma ferramenta de injeção de falhas que utiliza um objeto injetor de falhas em cada um dos processos que compõem o protocolo sob teste. AFIDS e a ferramenta são implementadas em C++ usando *sockets* em Linux.

2. Injeção de Falhas

Injeção de falhas visa validação de protocolos e auxílio ao projeto [ARL 90]. Na validação, são testados os métodos e mecanismos de tolerância a falhas através da introdução de falhas.

Os principais objetivos são:

- *a validação dos procedimentos de verificação* (por exemplo, conjuntos de teste) usados para revelar falhas durante todas as fases do processo de desenvolvimento;
- *a validação dos mecanismos de tolerância a falhas* (por exemplo, detecção, recuperação, etc) para alcançar a dependabilidade [LAP 85] do sistema na fase operacional.

A injeção de falhas envolve ainda dois aspectos da validação [KAR 94, ARL 90]:

- *previsão de falha* para avaliar a eficiência de mecanismos de manipulação de falhas fornecendo medidas como cobertura [MAR 93] e latência [ARL 90] da detecção de erros.
- *eliminação de falha* após a identificação de deficiências inesperadas (falhas de projeto) em mecanismos de tolerância a falhas.

À medida que a injeção de falhas é aplicada no projeto, ela pode auxiliar a refinar os mecanismos de teste e o próprio protocolo tolerante a falhas.

2.1 Classificação das técnicas de Injeção de Falhas

As técnicas de injeção de falhas podem ser classificadas em:

a) Injeção de Falhas Aplicadas a Modelos de Simulação

Em sistemas simulados, injeção de falhas é usada em vários níveis de abstração, como circuito, porta, RT (*register-transfer*), ou processos (alto nível) [KAR94]. Uma vantagem desta técnica é poder ser aplicada durante o desenvolvimento do sistema, o que facilita a detecção precoce de falhas de projeto. Outra vantagem é permitir alta controlabilidade e observalidade, através da seleção livre tanto do tempo e localização para a injeção de falhas, quanto dos pontos onde serão observadas as respostas do sistema. A principal desvantagem é o alto custo associado a simulação, o que coloca limitações práticas sobre a quantidade de hardware e software que pode ser modelada. Outra desvantagem está associada à transposição dos resultados da avaliação para a prática. Sistemas reais geralmente não se comportam sob falhas de forma idêntica a seus modelos. Uma ferramenta que implementa este tipo de injeção é FOCUS [CHO 92]. Muitos ambientes de simulação são validados por injetores específicos integrados ao código. No ADC [BAR 95], o ambiente de avaliação de protocolos de difusão confiável é um ambiente simulado e o injetor de falhas faz parte do ambiente.

b) Injeção de Falhas em Sistemas Físicos

Devido à complexidade envolvida, não é possível validar um sistema apenas pela injeção de falhas baseada em simulação. Deve-se então fornecer meios de injetar falhas em sistemas físicos. As técnicas de injeção em sistemas físicos podem ser subdivididas em [KAR 94]:

- a nível de pinos: falhas são injetadas fisicamente através de pinças de injeção (forçagem) ou posicionado o circuito sobre soquetes conectados a um injetor de falhas (inserção) [ARL 90, MAR 93A].
- através de distúrbios externos: falhas são injetadas através da exposição do circuito a íons pesados ou de distúrbios na voltagem de fontes de alimentação.
- *built-in*: mecanismos de injeção de falhas são incorporados a circuitos integrados. A injeção de falhas integra-se a técnicas de *scan chain* usadas em teste de circuitos.
- por software: falhas injetadas em um sistema modificam o estado do hardware/software do sistema sob controle do software, levando o sistema a agir como se falhas de hardware estivessem presentes [KAN95]. Basicamente consiste em interromper a execução da aplicação de alguma forma (geralmente inserindo um *trap* de software ou executando em modo *trace*) e executar o código do injetor de falhas. Esse código emula falhas de hardware pela inserção de erros em diferentes partes do sistema como registradores, memória, ou código da aplicação [CAR 95a]. Exemplos dessa classe de ferramentas são FIAT [SEG 88], EFA [ECH 92], SFI [ROS 93], DOCTOR [HAN 93], FINE [KAO 93], PFI [DAW 94], FERRARI [KAN 95], Xception [CAR 95a], ProFI [LOV 93].

As técnicas de injeção de falhas a nível de pinos e através de distúrbios externos tem a vantagem de injetar falhas de hardware reais. Mas podem danificar o componente sob teste [KAN 95, LOV 93], requerem o uso de hardware especial e as ferramentas de injeção de falhas são dedicadas a determinado sistema, e há dificuldade em controlar e observar as falhas dentro do processador [CAR 95a].

A injeção de falhas implementada por software tem como vantagens [CAR 95b]: baixo custo (não requer hardware dedicado), baixa complexidade e esforço de desenvolvimento, portabilidade aumentada, fácil expansibilidade para novos tipos de falhas, nenhum problema com interferências externas e nenhum risco de danificar o sistema destino (o sistema destino é o sistema sob teste, onde falhas são injetadas).

A injeção de falhas implementada por software tem problemas em modelar alguns tipos de falhas, como aquelas afetando a seção de controle do processador. A execução do injetor de falhas afeta as características de temporização do sistema, prejudicando o teste de funções de tempo críticas [KAR 94]. ORCHESTRA [DAW 96] aproveita características do Real Time Mach para minimizar o *overhead* da injeção de falhas no teste de protocolos tolerantes a falhas em sistemas de tempo real.

2.2 Injeção de Falhas e outras Técnicas de Validação

"A injeção de falhas pode complementar outras técnicas de validação [SEG 88, ARL 90, MAR 93A] como modelagem analítica e *error logging*. Modelagem analítica é extremamente difícil pois precisa de informação retirada da análise experimental [KAO 93] e a simplificação de assertivas, feitas de modo a fazer a análise tratável, reduz a utilidade dos resultados [KAN 95]. *Error logging* armazena informações sobre a ocorrência de erros no sistema em funcionamento, mas não pode ser empregada em sistemas que requerem extrema confiabilidade, onde teste após o uso do sistema é inapropriado [SEG 88, KAN 95].

Barton [BAR 90] mostra, através dos resultados de experimentos usando FIAT, que há um número limitado de manifestações de falhas nos níveis de abstração superiores ao nível que as falhas reais ocorrem. Isto possibilita que ferramentas de injeção de falhas possam realizar emulações de falhas a níveis de abstração mais altos que os das falhas reais.

3. Projeto de Ferramentas de Injeção de Falhas

São consideradas apenas ferramentas que manipulam mensagens. Oito ferramentas (FIAT [SEG 88], EFA [ECH 92], SFI [ROS 93], DOCTOR [HAN 93], PFI [DAW 94], CSFI [CAR 95], SockPFI [DAW 95], ORCHESTRA [DAW 96]) serviram de base para a determinação das características principais que devem ser consideradas no projeto de uma ferramenta de injeção de falhas, características estas que foram seguidas no desenvolvimento de AFIDS.

a) Interferência mínima sobre o sistema destino

A interferência da ferramenta de injeção de falhas sobre o sistema destino deve ser mínima, de modo a não prejudicar a precisão das medidas obtidas nos experimentos. Interferência mínima é essencial em sistemas de tempo real, onde as características temporais devem ser preservadas. SockPFI e ORCHESTRA aproveitam as características do sistema operacional de tempo real (Real Time Mach) para minimizar o impacto da ferramenta de injeção de falhas.

b) Automação dos experimentos

Esta característica visa acelerar o processo de injeção de falhas, e é baseada em: a) arquivos para a descrição dos experimentos de injeção de falhas, b) geração automática de casos de falhas e c) geração de workloads sintéticos. Todas as ferramentas citadas usam arquivos para descrição dos experimentos de injeção de falhas. A única ferramenta que gera automaticamente casos de falhas é EFA. Somente DOCTOR entre as oito ferramentas citadas gera workloads sintéticos.

c) Acesso ao código fonte do protocolo a ser testado no sistema destino

As únicas ferramentas que não precisam do código fonte do protocolo disponível para os experimentos são PFI, SockPFI e ORCHESTRA. É necessário o conhecimento do formato do pacote de mensagens do protocolo sob teste. SockPFI e ORCHESTRA precisam do acesso ao código objeto do protocolo sob teste.

d) Injeção determinística de falhas

A injeção determinística, em contraste com a randômica, possibilita guiar o sistema destino a estados "difíceis de alcançar" [DAW 94], reproduzir experimentos de injeção de falhas [ECH 92, CAR 95b] e alcançar uma alta cobertura de erros de projeto com um número menor de casos [ECH 91, ECH 94]. As ferramentas EFA, PFI, SockPFI, CSFI e ORCHESTRA fornecem esta característica.

e) Portabilidade

Várias ferramentas de injeção foram construídas especificamente para um sistema distribuído particular. SFI e DOCTOR foram projetadas para o sistema distribuído HARTS. CSFI foi implementado para uma rede de transputers, mas pode ser usado em outras plataformas com poucas modificações. EFA foi projetada e

implementada sobre o sistema operacional de tempo real A\ROSE. PFI foi inicialmente desenvolvida sobre o x-kernel rodando sobre Mach 3.0 e depois foi portado para o SunOS. SockPFI foi desenvolvida sobre Real Time Mach. ORCHESTRA roda em Real Time Mach, SunOS e Solaris.

f) Transparência para a aplicação

Há três formas de implementação do mecanismo de injeção de falhas:

- Adição de um objeto no código fonte: o pessoal de teste, ou a própria ferramenta de injeção, deve adicionar um objeto injetor no código fonte do protocolo tolerante a falhas.
- Colocação de uma camada de injeção entre duas camadas do sistema de comunicação a nível de sistema operacional: o pessoal de teste deve ter acesso ao formato dos pacotes das mensagens e ao núcleo do sistema operacional, como em EFA, CSFI e SFI. DOCTOR usa esta forma através do sistema operacional x-kernel rodando sobre HARTS.
- Colocação de uma camada de injeção entre duas camadas do sistema de comunicação a nível de usuário: o pessoal de teste deve ter acesso ao formato dos pacotes de mensagens e ao código objeto do aplicação. SockPFI e ORCHESTRA usam esta forma para aplicações que utilizam comunicação através de *sockets*. O código objeto do protocolo é religado com a biblioteca de injeção, que utiliza a biblioteca de *sockets* original. FIAT é implementado de forma semelhante.

4. AFIDS

A motivação para o desenvolvimento de AFIDS se deve principalmente à dificuldade enfrentada nas implementações de técnicas de tolerância a falhas, ADC [BAR 95] e FIX [TEI 95], recentemente concluídas. Nessas implementações, que abordam comunicação confiável e recuperação de processos em sistemas distribuídos, a maior dificuldade é determinar o comportamento sob falhas das técnicas implementadas. No sistema FIX foi previsto, mas ainda não implementado, um módulo de injeção de falhas acionado diretamente a partir de chamadas de sistema do Linux. Esse módulo forneceria primitivas que poderiam ser usadas para a construção de ferramentas de injeção de falhas em protocolos de recuperação de processos. No ADC adotou-se outra abordagem. Como o sistema de avaliação de protocolos de difusão roda em um ambiente simulado sob controle de um módulo central, as falhas são injetadas por esse módulo de acordo com uma probabilidade definida por um modelo de entrada. Entretando, apenas uma pequena classe de falhas é tratada. As duas soluções citadas para injeção de falhas são específicas para cada uma das implementações.

Então AFIDS pretende ser um *framework* para a construção destas ferramentas. Segundo [BOO 96]: "Através do uso de *frameworks* maduros, o esforço de desenvolvimento torna-se mais fácil, porque os principais elementos funcionais podem ser reutilizados".

4.1 Características de AFIDS

AFIDS visa possibilitar a construção de ferramentas de injeção de falhas implementada por software para sistemas distribuídos através do fornecimento de uma biblioteca de classes básica e genérica em C++, que disponibilize uma estrutura para suportar a geração de parâmetros de falhas, a injeção efetiva da falha e o seu controle e, finalmente, a coleta de dados dos experimento e a sua análise. Visa também permitir a validação de protocolos tolerantes a falhas para sistemas distribuídos, seja através de eliminação ou prevenção de falhas.

AFIDS está restrita à emulação de falhas de comunicação em sistemas distribuídos, suportando os seguintes tipos de falhas sobre mensagens: omissão de recebimento, omissão de envio, valor, queda de canal de comunicação, queda de processo, tempo e bizantina. Estas falhas são as mesmas suportadas por PFI [DAW 94]. AFIDS fornece flexibilidade na injeção de falhas pois permite a injeção em vários níveis do protocolo. A princípio pode ser utilizado em qualquer plataforma UNIX com o Compilador GCC da Free Software Foundation.

Atualmente AFIDS é um protótipo. Isto é enfatizado porque, segundo [BOO 96]: "Um *framework* só começa a alcançar maturidade após a sua aplicação em pelo menos três ou mais aplicações distintas".

4.2 Sistema Destino

Qualquer estação de trabalho executando sistemas operacionais UNIX pode suportar AFIDS, visto que ela usa C++ e *sockets* sem utilizar características específicas de algum sistema em particular. AFIDS está sendo implementada em PC's rodando o sistema operacional Linux.

AFIDS permite construir ferramentas para testar vários protocolos tolerantes a falhas. Inicialmente está sendo usada para construir uma ferramenta para a validação de um protocolo de comunicação de grupo tolerante a falhas chamado Newtop [EZH 95]. Este protocolo foi implementado em SunOS por Macedo [EZH 95]. A validação de Newtop irá auxiliar a refinar AFIDS.

Depois desta fase, AFIDS será usada para construir ferramentas para validar outros protocolos tolerantes a falhas. Já que AFIDS é orientado a objetos, é possível melhorar os mecanismos implementados em AFIDS através da modificação da estrutura de classes que os implementa.

4.3 Componentes de AFIDS

AFIDS fornece mecanismos para gerenciar os principais componentes das ferramentas de injeção de falhas implementada por software para sistemas distribuídos, como mostrado na figura 4.1 e na figura 4.2. Estas figuras detalham somente quatro processos e três nodos no sistema destino, mas AFIDS permite manipular o número total de processos e nodos necessários para a execução do protocolo tolerante a falhas sob teste. As seções 4.4.1 e 4.4.2 mostram dois projetos possíveis de duas ferramentas baseadas em AFIDS.

O arquivo *fault_parameters* corresponde ao conjunto *F* (falhas) nos conjuntos FARM (Falhas, Ativações, Resultados e Medidas) [ARL 90]. As ações de envio e recebimento dos objetos *injector* correspondem ao conjunto *A* (Ativações). O arquivo *readouts_data* corresponde ao conjunto *R* (Resultados). O arquivo *measures_data* corresponde ao conjunto *M* (Medidas).

O hospedeiro (*host*) é uma máquina que controla o processo de injeção de falhas. Os objetos *manager*, *generator*, *controller*, *collector*, *analyser* e *name_server* e os arquivos *fault_parameters*, *standard_data*, *readouts_data* e *measures_data* são colocados no hospedeiro. O sistema destino (*target system*) corresponde aos nodos onde os processos do protocolo tolerante a falhas rodam. Os objetos *injector* e o arquivo *faults_base* são colocados no sistema destino.

4.3.1 Objetos

Os principais objetos de AFIDS são:

a) *manager* gerencia os outros componentes da injeção de falhas.

b) *generator* gera um arquivo de parâmetros de falhas (*fault_parameters*).

c) *name_server* relaciona os objetos *injector*, os processos, os nodos e grupos de processos se houver. Este relacionamento é realizado durante a geração do arquivo de parâmetros de falhas. O arquivo *name_base* é utilizado para armazenar os identificadores. A seção 4.3.3 ilustra em detalhes o objeto *name_server*.

d) *controller* controla a conexão entre os objetos *injector*, quando eles são criados ou destruídos, e envia aos objetos *injector* a informação dos parâmetros de falhas.

e) *collector* coleta os dados padrões (a partir da execução do protocolo sem injeção de falhas) colocando os dados no arquivo *standard_data*. Posteriormente coleta os resultados da injeção de falhas (a partir da execução do protocolo com a injeção de falhas, colocando os dados no arquivo *readouts_data*). Controla também a conexão com os objetos *injector* quando eles são criados ou destruídos.

f) *injector* recebe os parâmetros das falhas que este objeto deve injetar. Estes parâmetros são armazenados em uma base de falhas (*faults_base*) pelo objeto *injector*, que é sempre consultado para que a falha seja injetada no seu devido tempo. Ele também se comunica com os objetos *controller* e *collector* para informá-los sobre a sua criação ou destruição.

g) *analyser* avalia o arquivo *standard_data* e o arquivo *readouts_data* de modo a obter medidas de dependabilidade [LAP 85], que são colocadas no arquivo *measures_data*. As medidas de dependabilidade, como por exemplo confiabilidade e disponiblidade, devem ser determinadas pelo pessoal de teste.

Para a comunicação entre os objetos de AFIDS podem ser utilizadas quaisquer interfaces de comunicação, seja *sockets*, RPC, PVM ou alguma outra plataforma de comunicação distribuída.

4.3.2 Arquivos

Os principais arquivos de AFIDS são:

a) *fault_parameters*: É gerado pelo objeto *generator* ou é escrito pelo pessoal de teste. Armazena os parâmetros das falhas a serem injetadas. A seção 4.3.4 ilustra em detalhes o arquivo *fault_parameters*.

b) *faults_base*: Armazena os parâmetros das falhas que serão injetadas por um objeto *injector* em particular. Ou seja, cada objeto *injector* deve possuir um arquivo *faults_base*. Este arquivo elimina a necessidade de comunicação adicional entre o hospedeiro e o sistema destino após a criação do objeto *injector*.

c) *name_base*: Armazena os identificadores dos processos, grupos, nodos e objetos *injector*. É utilizado pelo objeto *name_server*.

d) *standard_data*: Armazena os dados (valores de variáveis e mensagens) resultantes da execução do protocolo tolerante a falhas sem a injeção das falhas.

e) *readouts_data*: Armazena os dados (valores de variáveis e mensagens) resultantes da execução do protocolo tolerante a falhas com a injeção das falhas.

f) *measures_data*: Armazena as medidas de dependabilidade resultantes da análise dos arquivos *standard_data* e *readouts_data* pelo objeto *analyser*. Exemplos de medidas são a cobertura e a latência dos mecanismos tolerantes a falhas.

Para acelerar o acesso aos dados, os arquivos *name_base* e *faults_base* podem ser implementados em estruturas de dados armazenadas na memória principal no lugar de armazená-los na memória secundária.

4.3.3 O Objeto *Name_Server*

O objeto *name_server* é um servidor de nomes que armazena identificadores (no arquivo *name_base*) para os objetos *injector* de modo a facilitar a relação entre o nodo, grupo ou processo onde uma falha deve ser injetada. É composto pela seguinte estrutura de dados:

- Identificação do objeto *injector* (*InjectorId*): uma cadeia de caracteres que identifica o objeto *injector*.
- Endereço do objeto *injector* (*InjectorAddress*): o endereço IP do nodo.
- Nodos responsáveis pelo objeto *injector* (*NodesList*): os nomes dos nodos.
- Grupos responsáveis pelo objeto *injector* (*GroupsList*): os identificadores dos grupos que o objeto *injector* é responsável.
- Processos responsáveis pelo objeto *injector* (*ProcessList*): os identificadores dos processos que o objeto *injector* é responsável.

As operações básicas são (baseadas no modelo SNS (*Simple Name Service*) [COU94]):

- *Bind* (*InjectorId, InjectorAddress, NodesList, GroupsList, ProcessList*) ({*Success, AlreadyExist*}: cria um entrada no *name_server* que armazena as informações sobre cada objeto *injector*.
- *LookupId* (*InjectorId*) ({*Success, NotFound*}: procura uma identificação de um objeto *injector* e retorna os atributos se a identificação é encontrada.
- *LookupAttributes* (*InjectorAddress, NodesList, GroupsList, ProcessList*) ({*Success, NotFound*}: procura uma entrada com os atributos e retorna a identificação do objeto *injector* correspondente.
- *Unbind* (*InjectorId*) ({*Success, NotFound*}: deleta uma entrada no *name_server*.

4.3.4 O arquivo *fault_parameters*

Como exemplo de parâmetros de falhas do arquivo *fault_parameters* podem ser citados:

- Tempo da Injeção de Falhas: Indica em qual mensagem a falha será injetada. É usado o número do pacote da mensagem.
- Localização das Falhas: cabeçalho ou os dados da mensagem.
- Duração das Falhas: Pode ser temporário, permanente ou intermitente. Para falhas temporárias e permanentes, a duração é o número de pacotes de mensagens. Para falhas intermitentes, é um número obtido de uma função de distribuição de probabilidade ou um número dado pelo pessoal de teste.
- Nodo onde as falhas serão injetadas: Nome do nodo ou o endereço IP do nodo.
- Processo onde as falhas serão injetadas: É alguma identificação do processo.
- Grupo de processos onde as falhas serão injetadas: É a identificação do grupo de processos que receberão a injeção das falhas. É importante em protocolos que envolvem grupos de processos (por exemplo, comunicação em grupo e replicação de dados).
- Tipo de Falha que será usado: Tipo de falhas de comunicação (todas as falhas implementadas pelas funções de envio e recebimento dos objetos *injector*).
- Padrão de Falha: É uma cadeia de caracteres que irá ser usada para modificar o conteúdo das mensagens.

4.4 Opções de projeto de ferramentas baseada em AFIDS

As seções 4.4.1 e 4.4.2 apresentam duas opções de projeto básicas para o desenvolvimento de novas ferramentas de injeção baseadas em AFIDS. Estas opções são apenas duas entre muitas. Outras opções podem ser imaginadas através da manipulação dos objetos e os arquivos de AFIDS de acordo com os objetivos do desenvolvedor.

4.4.1 Com o componente de injeção em cada processo do protocolo sob teste

A organização dos componentes da ferramenta baseada em AFIDS esboçada na figura 4.1 não foi utilizada por nenhuma das ferramentas analisadas (FIAT, EFA, SFI, DOCTOR, CSFI, PFI, SockPFI, ORCHESTRA).

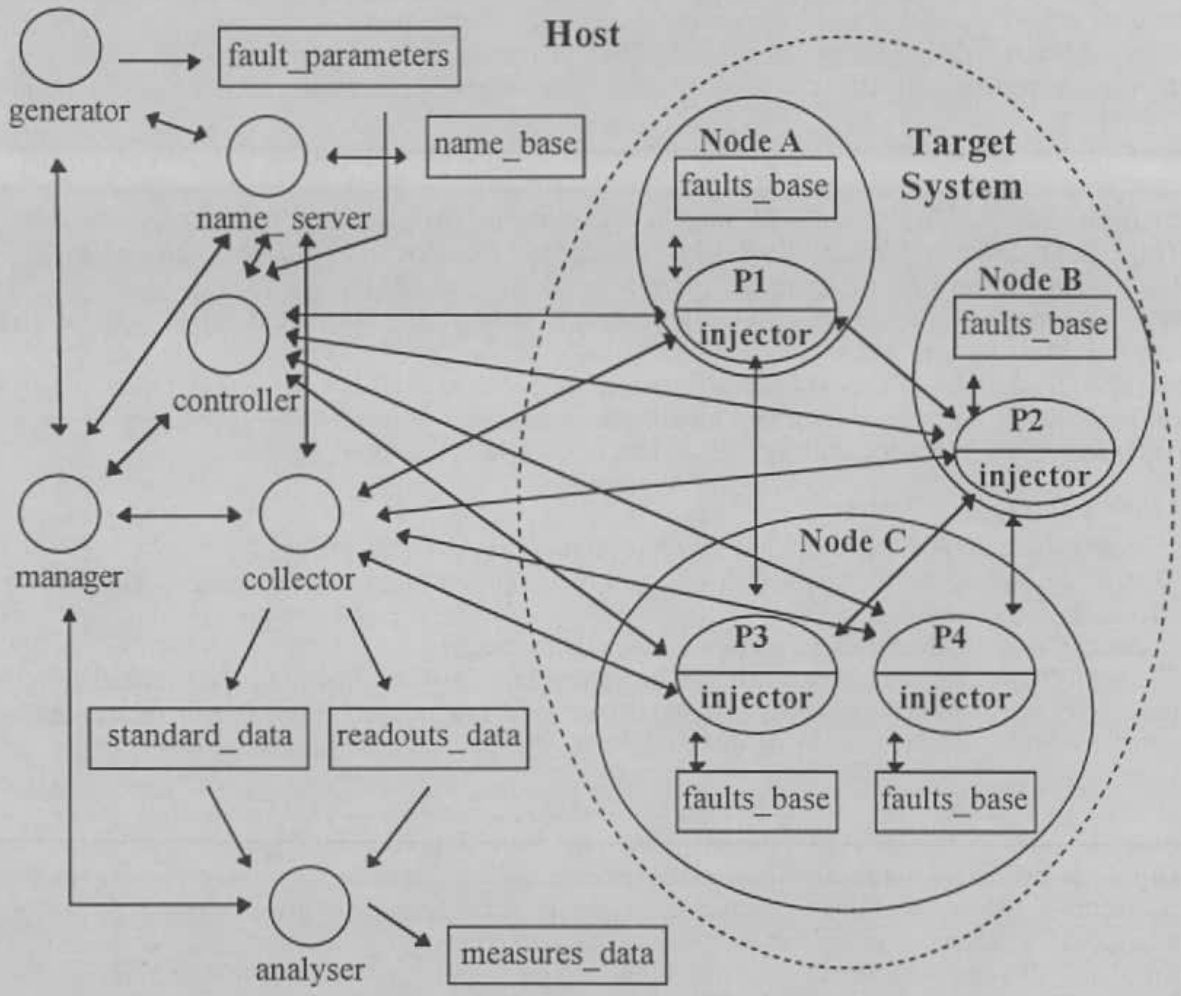

Figura 4.1 - Objetos principais de AFIDS-ip, com um objeto *injector* por processo

Esta forma de organização será referida no texto como AFIDS-ip (ferramenta baseada em AFIDS com 1 objeto *injector* por processo do protocolo sob teste) e acopla um objeto injetor dentro do código fonte do(s) processo(s) do protocolo a ser testado. Mas a idéia de acoplar um objeto injetor em cada processo do protocolo sob teste é similar àquela de FIAT, onde o objeto injetor era ligado ao processo do protocolo. E similar à implementação de SockPFI e ORCHESTRA, onde o código objeto do protocolo sob teste é ligado a uma biblioteca de *sockets* alterada (biblioteca responsável pela injeção das falhas), que por sua vez usa a biblioteca de *sockets* original.

Neste projeto de AFIDS-ip, o código do protocolo tolerante a falhas a ser validado tem que estar disponível para ser modificado da seguinte forma:

- Cada processo do protocolo tolerante a falhas que envia ou recebe mensagens incorpora um objeto *injector* , que será colocado no início do processo do protocolo tolerante a falhas.
- As funções de envio e recebimento de cada processo do protocolo tolerante a falhas devem ser substituídas por uma função membro de envio ou recebimento do objeto *injector*. A função membro do objeto *injector* encapsula a função original do processo do protocolo de modo a injetar falhas.

4.4.2 Com o componente injetor entre duas camadas do sistema de comunicação

Devido à sua natureza orientada a objetos, é possível estender AFIDS para suportar a organização em camadas, onde o componente injetor (objeto *injector*) é colocado entre duas camadas de comunicação, da forma que está esboçada na figura 4.2.

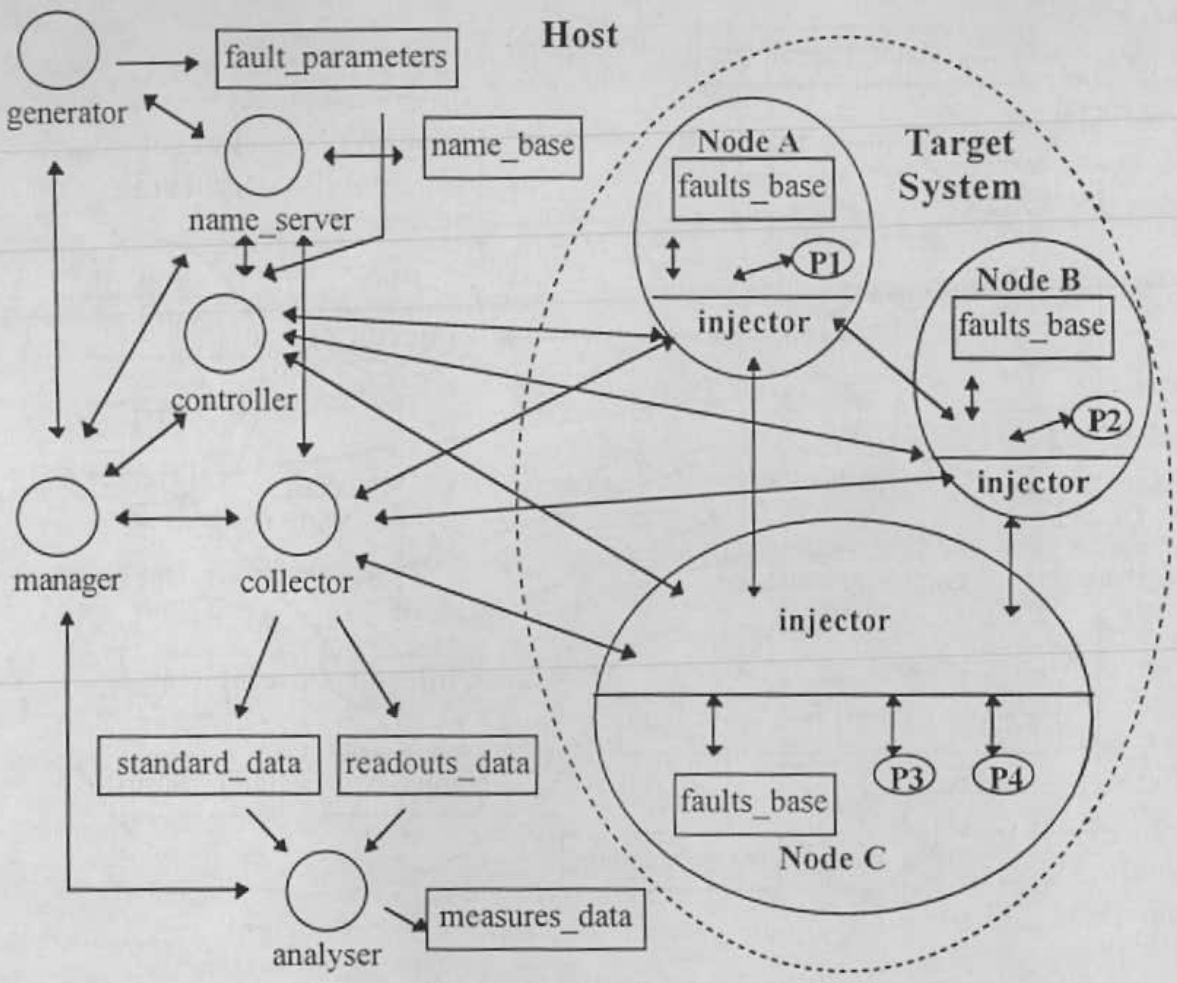

Figura 4.2 - Objetos principais de AFIDS-in (1 objeto *injector* por nodo).

Nesta forma de organização há a necessidade de acesso ao kernel para incluir a camada de injeção entre duas camadas de comunicação, como em EFA e CSFI. Ou então utilizar o x-kernel a nível de S.O. e incluir a camada de injeção dentro do sistema de camadas de comunicação do x-kernel, como em DOCTOR. Uma outra forma, seria utilizar o filtro de pacotes existente no SunOS (NIT [SUN 87]). A forma da figura 4.2 será referida no texto como AFIDS-in (ferramenta com 1 objeto *injector* por nodo).

5. Implementação baseada na idéia de AFIDS-ip

A implementação inicial de AFIDS, que será usada para avaliar o protocolo NewTop, considera o modelo de uma ferramenta com 1 objeto injector por processo por ser mais simples de implementar do que AFIDS-in. Como consequência, a biblioteca AFIDS pode ser construída mais rapidamente, permitindo acelerar o entendimento dos procedimentos de injeção. Apesar de ser necessária a intromissão no código, possui flexibilidade para possibilitar experimentos de injeção de falhas praticamente em qualquer nível de abstração no sistema de comunicação. Por exemplo, um protocolo de comunicação de grupo que utilize sockets pode ser validado no mais alto nivel de abstração (primitivas de comunicação multicast fornecidas pelo protocolo) ou no mais baixo nível de abstração (primitivas de comunicação a nível de sockets).

Uma implementação futura de AFIDS-in irá reutilizar as classes de AFIDS-ip e irá realimentar a biblioteca AFIDS com novas classes e métodos genéricos. A implementação de AFIDS-in diminuiria a carga existente em AFIDS-ip, onde há um objeto *injector* em cada processo do protocolo. Isto porque haveria apenas um objeto *injector* por nodo. Mais ainda, com AFIDS-in, há a possibilidade de não ser necessária a intromissão no código fonte do protocolo, por exemplo, de forma similar a PFI ou EFA.

5.1 Componentes de AFIDS-ip

A Figura 5.1 refina a Figura 4.1 criando o objeto *controller-collector* através da agregação dos objetos *controller* e *collector*. Assim ocorre uma diminuição na comunicação entre o sistema destino e o hospedeiro.

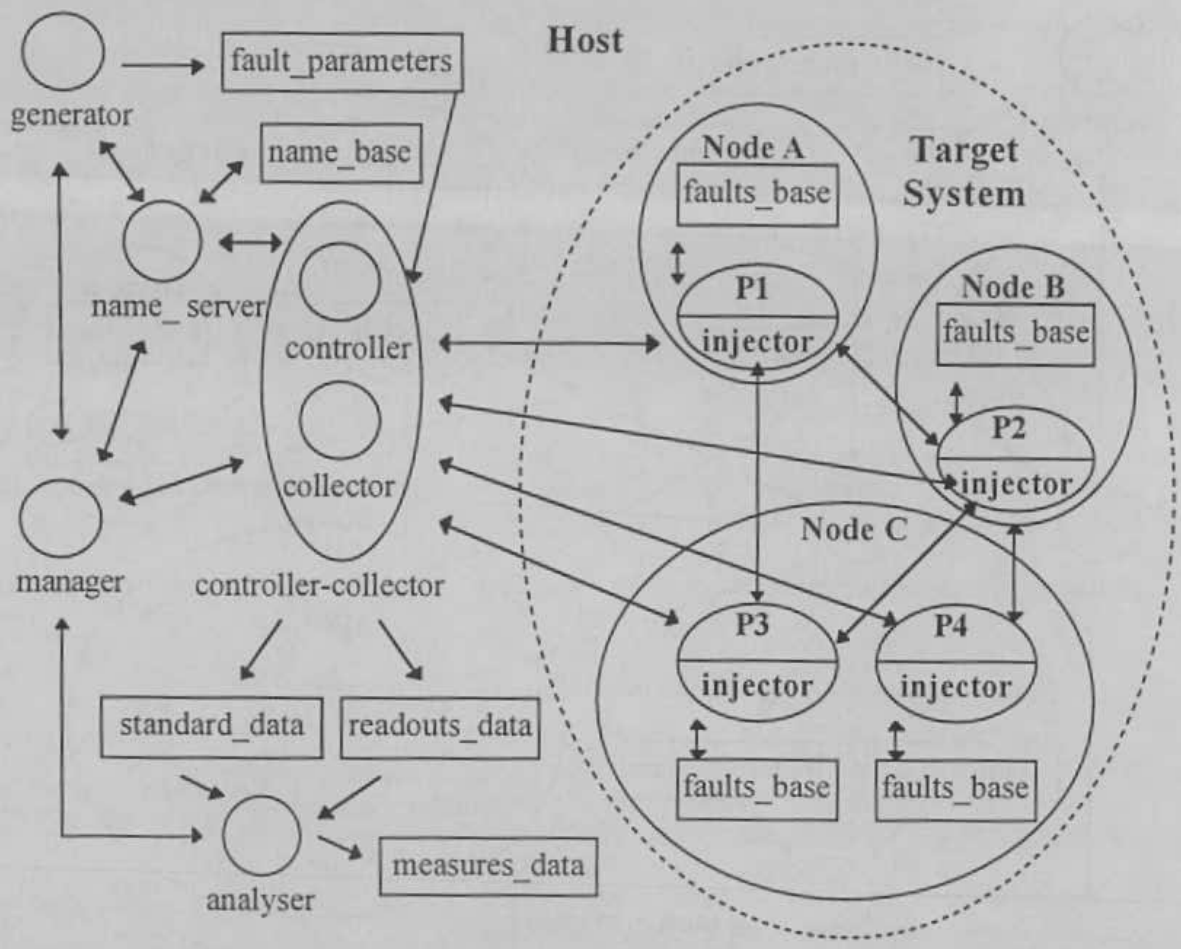

Figura 5.1 - AFIDS-ip modificada

AFIDS-ip tem duas caracteristicas importantes: gerenciamento flexível da criação e destruição de processos e comunicação reduzida entre os objetos *controller-collector* e *injector*.

O gerenciamento flexível da criação e destruição dos processos do protocolo tolerante a falhas, permitindo que os objetos *injector* informem sobre a suas próprias criações e destruições ao objeto *controller-collector*, possibilita que os objetos *injector* sejam criados e destruídos dinamicamente durante a execução do protocolo tolerante a falhas.

A comunicação é reduzida entre o objeto *controller-collector* e os objetos *injector*, porque, quando um objeto *injector* é criado, o objeto *controller-collector* envia a ele os parâmetros das falhas e a identificação única gerada pelo objeto *name_server*. Então o objeto *injector* armazena no arquivo *faults_base* os parâmetros de falhas recebidos e que serão injetadas por ele. Depois disso, os objetos *injector* não precisam esperar por uma mensagem do objeto *controller-collector* a respeito das falhas a serem injetadas, reduzindo assim a comunicação entre eles.

O formato dos pacotes de comunicação entre os componentes de AFIDS-ip estão descritos no Anexo.

5.2 O Cenário de Execução de AFIDS-ip

Um cenário tipico da injeção de falhas utilizando a arquitetura da figura 5.1 é:

1. O objeto *manager* é criado. O construtor do objeto *manager* instancia os objetos *name_server*, *generator*, *controller-collector* e *analyser*.

2. Uma função membro do objeto *manager* pergunta ao usuário sobre quatro opções:

- Iniciar o processo de injeção de falhas. Esta opção supõe que um arquivo *fault_parameters* esteja disponivel. Vá para o Passo 3.
- Gerar o arquivo *fault_parameters*. O objeto *generator* interage com o usuário e cria o arquivo *fault_parameters*. Após os parâmetros de falhas serem gerados, o objeto *generator* invoca o objeto *name_server* que armazena os identificadores no arquivo *name_base*. Vá para o Passo 2.
- Chamar o objeto *analyser* que analisa os arquivos *standard_data* e *readouts_data* e gera o arquivo *measures_data*. Vá para o Passo 2.
- Terminar o processo de injeção de falhas. Vá para o Passo 7.

3. O objeto *controller-collector* inicia o gerenciamento de sinais (mensagens) dos objetos *injector*. Ele fica esperando por dois possíveis sinais provenientes dos objetos *injector*:

- No recebimento de um sinal de criação, ele estabelece a conexão para o novo objeto *injector* e requisita ao objeto *name_server* uma identificação para o objeto *injector* recém-criado. Além disso, o objeto *controller-collector* lê o arquivo *fault_parameters* e obtém os parâmetros de falhas que o objeto *injector* recém-criado deve injetar. Então o objeto *controller-collector* envia ao objeto *injector* os parâmetros de falhas de interesse do novo objeto *injector* e um identificador único para o objeto *injector*.
- No recebimento de um sinal de destruição, ele fecha a conexão com o objeto *injector*.

4. O objeto *controller-collector* então fica esperando pelos dados dos objetos *injector* para gerar o arquivo *readouts_data*. Para cada criação de um processo do protocolo tolerante a falhas, cada construtor do objeto *injector* envia um sinal de criação para o objeto *controller-collector*. Para cada destruição de um processo do protocolo tolerante a falhas, cada destrutor do objeto *injector* envia um sinal de destruição para o objeto *controller-collector*.
5. O protocolo tolerante a falhas é executado e os dados da execução são coletados pelo objeto *controller-collector*, que gera o arquivo *readouts_data*.
6. Quando o objeto *controller-collector* percebe que a execução do protocolo tolerante a falhas terminou (discutido na seção 5.3), ele envia um sinal (mensagem) para o objeto *manager* informando-o sobre o término da execução do experimento de injeção. Vá para o Passo 2.
7. O objeto *manager* é destruído. O destrutor do objeto *manager* destrói os objetos *name_server, generator, controller-collector* e *analyser*.

5.3 Detecção da Terminação da Execução do Protocolo Tolerante a Falhas

De modo a permitir que o objeto *controller-collector* saiba do término da execução do protocolo tolerante a falhas, seja sem ou com injeção de falhas, o seguinte procedimento é aplicado:

a) São usadas duas variáveis auxiliares no objeto *controller-collector*:
- int creationCounter; /* É um contador de objetos *injector* criados. Inicialmente é inicializado com 0. */
- char someInjectorCreated; /* É uma variável lógica que indica se algum objeto *injector* já foi criado (1) ou não (0). Inicialmente é inicializado com 0. */

b) Cada vez que *controller-collector* recebe um sinal de criação de algum objeto *injector*, creationCounter é aumentado de 1 unidade. Na primeira vez que creationCounter é incrementado, a variável lógica someInjectorCreated é setada para 1.
c) Cada vez que o objeto *controller-collector* recebe um sinal de destruição de algum objeto *injector*, creationCounter é diminuído de 1 unidade.
d) Quando someInjectorCreated for igual a 1 e creationCounter for igual a 0, então o objeto *controller-collector* sabe que a execução do protocolo já terminou e retorna o controle do experimento para o objeto *manager*.

6. Utilização da Ferramenta

Para a utilização de AFIDS-ip (ferramenta baseada em AFIDS com 1 objeto *injector* por processo do protocolo sob teste), deve-se seguir os seguintes passos: 1) decidir em qual nível no sistema de comunicação as falhas serão injetadas, 2) construir um arquivo de parâmetros de falhas, 3) inserir um objeto *injector* em cada processo do protocolo tolerante a falhas, 4) encapsular as funções de envio e recebimento do protocolo dentro dos métodos da classe de injeção, 5) executar os experimentos e 6) analisar os dados coletados. Este capítulo discute cada um destes passos.

6.1 Escolha do nível do sistema de comunicação para a injeção de falhas

Em protocolos ou sistemas onde se tem acesso ao código fonte, e cujas implementações possuem mais de um nível de abstração, é possível escolher em qual deles a injeção de falhas será realizada.

Por exemplo, em um protocolo de comunicação de grupo confiável, que é implementado usando *sockets*, há no mínimo dois níveis de abstração em relação à comunicação:

- o nível mais baixo correspondente ao nível de *sockets*, onde as funções de envio e recebimento entre os processos são executadas via chamadas a rotinas de biblioteca de *sockets*. Neste caso, a injeção de falhas poderia ser implementada através do encapsulamento das funções de envio e recebimento de *sockets* (*read*, *recv*, *write*, *send*, etc).
- o nível do protocolo correspondente à utilização de grupos. A injeção de falhas poderia ser implementada através do encapsulamento das funções responsáveis pelo envio e recebimento de mensagens de grupos de processos. Por exemplo, dois grupos, A e B, e as funções de envio (A.send() e B.send()) e recebimento (A.receive() e B.receive()).

Esta possibilidade de escolha auxiliaria na detecção de possíveis erros de projeto/implementação do protocolo nos diferentes níveis de abstração.

6.2 Construção de um arquivo de parâmetros de falhas

Estudos para a geração de parâmetros de falhas, por exemplo, baseados no código fonte do protocolo [ECH 91] ou em modelos formais (por ex. redes de Petri) [ECH 94], preocupam-se em aumentar a cobertura dos experimentos de injeção de falhas com um número mínimo de casos de falhas (usa-se o conceito de cobertura como definido por Martins [Mar 93]). O estado atual de implementação AFIDS-ip se preocupa apenas em especificar as falhas. A utilização de métodos sofisticados é deixado como trabalho futuro.

Para exemplificar, admita um protocolo de comunicação de grupo confiável com os seguintes nodos (Sirius e Rigel), grupos (A, B e C) e processos (A1, A2, A3, B1, B2, C1 e C2), especificados no arquivo de parâmetros de falhas (*fault_parameters*):

```
Node Sirius{
   Groups (processes): A (A1, A2, A3), B (B1, B2);
   Injector Id : IA1, IA2, IA3, IB1, IB2;
};
Node Rigel{
   Groups (processes): C (C1, C2);
   Injector Id : IC1, IC2;
};
```

Os parâmetros de falhas , ainda no arquivo *fault_parameters*, são especificados da seguinte forma:

```
FaultParameters {
   Fault 0{
         InjectionTime: 10;
         FaultLocation: header;
         FaultDuration: temporary (5);
         Node: Sirius;
         Process: A1;
         FaultPattern: "lixo no cabeçalho";
   }
   Fault 1{
         InjectionTime: 30;
         FaultLocation: header;
         FaultDuration: intermitent (10);
         Node: Sirius;
         Process: B2;
         FaultPattern: "lixo no cabeçalho";
   }
   Fault 2{
         InjectionTime: 20;
         FaultLocation: data;
         FaultDuration: permanent;
         Node: Rigel;
         Group: C;
         Process: C1, C2;
         FaultPattern: "lixo nos dados";
   }
};
```

Cada falha tem um número (Fault **number**) correspondente à posição em um vetor de falhas. O tempo de injeção (InjectionTime: **number**) corresponde a um número do pacote de mensagem do protocolo sob teste. A localização da falha (FaultLocation: **header** | **data**) indica se a falha será injetada no cabeçalho ou na área de dados do pacote de mensagem.

A duração da falha (FaultDuration) pode ser temporária, permanente ou intermitente. A falha temporária (temporary (**number**)) é injetada desde o pacote definido em InjectionTime até o pacote definido em **number**. A falha permanente (permanent) é injetada desde o pacote definido por InjectionTime até o fim do experimento). A falha intermitente (intermitent (**number**)) ocorre periodicamente desde InjectionTime a intervalos definidos por **number**.

O nodo (Node: **name**) define qual o nodo que a falha será injetada. O grupo (Group: **name**) define em qual grupo de processos a falha será injetada. A lista de processos (Process: **process_list**) define os processos onde a falha será injetada. O padrão de falha (FaultPattern: **pattern**) define qual a cadeia de caracteres que será utilizada para realizar a injeção da falha.

Com a criação do arquivo de parâmetros de falhas (*fault_parameters*), o objeto *generator* envia os identificadores no servidor de nomes (objeto *name_server*).

A especificação acima será utilizada nas seções seguintes. A definição da gramática e a implementação da linguagem para esta especificação é um trabalho futuro, os testes atuais definem os parâmetros estaticamente no código.

6.3 Inserção de um objeto *injector* em cada processo do protocolo

Para possibilitar a utilização dos parâmetros de falhas especificados na seção 6.2, os processos (A1, B2, C1 e C2) do protocolo precisam inserir um objeto *injector* no início da função principal. Exemplo usando o processo A1:

```
main ()
{
   AFIDSipInjector IA1;
   Corpo original da mensagem
}
```

6.4 Encapsulamento das funções de envio e recebimento do protocolo

As funções de envio e recebimento destes processos devem ser encapsuladas com as funções de envio e recebimento do objeto *injector* inserido. Exemplo usando o processo A1:

```
main ()
{
   AFIDSipInjector A1Injector;
   ...
   A1Injector.send (função de envio original);
   ...
   A1Injector.receive (função de recebimento original);
   ...
}
```

A identificação para o objeto A1Injector irá ser recebida pelo processo A1 quando o construtor de A1Injector enviar o sinal de criação ao objeto *controller*. Com o recebimento do sinal de criação, o objeto *controller* consulta o objeto *nameServer* e percebe que a identificação é IA1 e envia para objeto *injector*. Desta forma a questão da identificação de objetos *injector*, nodos, processos e grupos é resolvida.

6.5 Execução dos experimentos

Detalhes da execução de AFIDS-ip podem ser encontrados na seção 5.2. Pri-meiramente é realizada a execução do protocolo sem a injeção de falhas e o arquivo *standard_data* é gerado. Para uma melhor escolha dos parâmetros de falhas, pode-se avaliar o arquivo *standard_data* e então modificar os parâmetros de falhas especificados no arquivo *fault_parameters*. Então se procede a execução com a injeção efetiva dos parâmetros de falhas, o que irá gerar o arquivo *readouts_data*.

O estado atual da implementação simplesmente lista as mensagens trocadas entre os processos que possuem os objetos *injector* inseridos. Isto para ambos os arquivos *standard_data* e *readouts_data*. O aperfeiçoamento desta implementação é deixada como trabalho futuro.

6.6 Análise dos dados coletados

A análise automática dos arquivos *standard_data* e *readouts_data* para gerar um arquivo com medidas comparativas (arquivo *measures_data*) é uma tarefa não trivial e também é deixada como sugestão de trabalho futuro.

O estado atual da implementação simplesmente não faz a análise automática. A análise dos arquivos *standard_data* e *readouts_data* é realizada de forma manual pelo pessoal de teste que estiver utilizando a ferramenta.

7. Conclusão

AFIDS visa fornecer uma biblioteca orientada a objetos para construir ferramentas de injeção de falhas implementada por software em sistemas distribuídos. AFIDS capacita construir rapidamente novas ferramentas com mecanismos melhorados ou adicionados. Esta característica permite otimizar, passo a passo, a organização e a estrutura de AFIDS, visando minimizar, a cada novo experimento, a interferência da ferramenta de injeção de falhas sobre o sistema destino.

Entre as mais recentes ferramentas para injetar falhas implementadas por software em sistemas distribuídos, somente PFI e ORCHESTRA executam em estações de trabalho rodando UNIX (SunOS e

Solaris). AFIDS é mais uma opção para injetar falhas em protocolos tolerantes a falhas em sistemas operacionais UNIX.

Sendo construída usando C++ e *sockets* sobre Linux, sem o uso de características específicas de Linux, é fácil portar AFIDS para outros sistemas operacionais UNIX compatíveis. Mais ainda, a interferência sobre o sistema destino é reduzida pela separação entre hospedeiro e sistema destino, da mesma forma que as ferramentas de injeção existentes, e por não usar nenhuma aplicação ou sistema adicional para injetar falhas. É possível até mesmo utilizar AFIDS em outras plataformas que não utilizem *sockets*, visto que a forma de comunicação dos componentes de AFIDS é uma opção do projetista da ferramenta.

A implementação de AFIDS-ip mostra que a construção de ferramentas de injeção de falhas é uma tarefa que necessita de muita articulação dos projetistas para a definição de uma arquitetura de ferramenta ideal para os objetivos deles. E tal tarefa está sendo facilitada com o desenvolvimento de AFIDS, que se apresenta em constante evolução.

Particularmente, na UFRGS, alguns trabalhos sobre protocolos tolerantes a falhas em sistemas distribuídos sobre sistemas operacionais UNIX (Linux, SunOS) têm sido desenvolvidos. Esses trabalhos envolvem replicação de dados, recuperação de processos e protocolos de difusão. AFIDS será usado para validar estes protocolos. Entretanto, de modo a refinar a arquitetura, o protocolo de comunicação de grupo tolerante a falhas Newtop [EZH 95] será o primeiro a ser validado.

EFA validou protocolos de mascaramento de falhas, concordância e ordenação, PFI validou protocolos de comunicação (TCP e GMP) e DOCTOR validou algoritmos de diagnóstico distribuído. No momento, AFIDS aborda somente o modelo de falhas de comunicação, que é o bastante para validar uma vasta gama de protocolos tolerantes a falhas em sistemas distribuídos baseados em *broadcast* ou *multicast*. Entretanto, para tornar a arquitetura mais abrangente, seria conveniente adicionar modelos de falhas de processador e memória, de modo a suportar o teste de algoritmos distribuídos, que não podem ser testados somente através do uso do modelo de falhas de comunicação.

Anexo: Formato dos Pacotes de Mensagens entre os Componentes de AFIDS-ip

Este anexo apresenta o formato dos pacotes das mensagens que são trocadas entre os componentes de AFIDS-ip. Todos os formatos de pacotes de mensagens definidos a seguir utilizam a estrutura header. A constante MAXLABELSIZE indica o tamanho máximo dos identificadores utilizados. O campo targetprocess indica qual o processo que receberá a mensagem. O campo sourceprocess indica o processo que envia a mensagem. O campo label é um número de sequência da mensagem. O formato em C++ é:

```
struct header
{
    char targetprocess[MAXLABELSIZE];
    char sourceprocess[MAXLABELSIZE];
    long label;
};
```

Formato dos Pacotes na Comunicação entre os Objetos *injector* e o *controller-collector*

a) Do objeto *injector* para o controller-collector na criação de *injector*:

O campo messageType é inicializado com 0 para identificar a mensagem como um sinal de criação do objeto *injector*. O campo nodo identifica qual o nodo que se encontra o processo que instanciou o objeto *injector*. O campo plabel identifica o processo que instanciou o objeto *injector*. O formato em C++ é:

```
struct InjectorCreationMessage
{
  header h;
  char messageType;
  char nodo[MAXLABELSIZE];
  char plabel[MAXLABELSIZE];
};
```

b) Do objeto *injector* para o *controller-collector* na destruição de *injector*:

O campo messageType é inicializado com 2 para identificar a mensagem como um sinal de destruição do objeto *injector*. O campo ilabel identifica o objeto *injector* que enviou a mensagem. O formato em C++ é:

```
struct InjectorDestructionMessage
{
  header h;
  char messageType;
```

```
  char ilabel[MAXLABELSIZE];
};
```

c) Do objeto *injector* para o *controller-collector* no envio de dados durante a Execução do Protocolo:

Em CollectedDataMessage, o campo messageType é inicializado com 4 para identificar a mensagem como uma mensagem normal do protocolo sob teste, ou com 8 para identificar o valor de alguma variável. O campo ilabel identifica o objeto *injector* que enviou a mensagem. O campo contentsSize indica o tamanho da mensagem normal do protocolo se messageType for 4, ou indica o tamanho do valor da variável se messageType for 8. O campo variableName armazena o nome da variável. O campo messageContents contém o conteúdo da mensagem normal do protocolo ou o valor da variável (transformada em uma *string*). O formato em C++ é:

```
struct CollectedDataMessage
{
  char messageType;
  char ilabel[MAXLABELSIZE];
  int contentsSize;
  char variableName[MAXLABELSIZE];
  char *messageContents;
};
```

d) Do objeto *controller-collector* para o *injector* na criação de *injector*:

Em FParam, o campo injectionTime (número de sequência das mensagens do protocolo sob teste) indica em qual mensagem será aplicada a injeção. O campo faultLocation indica se a falha será aplicada no cabeçalho ou na área de dados da mensagem. O campo faultDuration (quantidade de pacotes de mensagens do protocolo sob teste) indica a duração da aplicação das falhas. O campo faultType indica qual o tipo de falha de comunicação a ser injetada: 0 - omissão de recebimento, 1 - omissão de envio, 2 - valor, 3 - queda de canal de comunicação, 4 - queda de processo, 5 - queda de nodo e 6 - temporização. O campo patternSize indica o tamanho do campo faultPattern. O campo patternStart indica a partir de qual posição o padrão de falhas (campo faultPattern) será instalado. O campo faultPattern é uma cadeia de caracteres que irá ser usada para modificar o cabeçalho ou os dados das mensagens. O formato em C++ é:

```
struct FParam
{
  long injectionTime;
  char faultLocation;
  int faultDuration;
  long faultType;
  int patternSize;
  int patternStart;
  char *faultPattern;
};
```

Em faultParametersMessage, o campo messageType é setado para 16, indicando que a mensagem leva parâmetros de falhas. O campo ilabel armazena o identificador único do objeto *injector* que é obtido do objeto *name_server*. O campo faultsQuantity indica a quantidade de parâmetros de falhas que estão sendo enviados. O campo faultsList é a lista de parâmetros de falhas a serem injetadas. O formato em C++ é:

```
struct faultParametersMessage
{
  header h;
  char messageType;
  char ilabel;
  int faultsQuantity;
  FParam **faultsList;
};
```

Formato dos Pacotes na Comunicação entre os Objetos *controller-collector*, *generator* e *name_server*

a) Do objeto *generator* para o *name_server* na geração do arquivo *fault_parameters*:

O campo messageType é setado para 10 para indicar que nameSetMessage é uma mensagem de atribuição de um nome ao servidor de nomes. O campo nodo é o identificador do nodo onde se localiza o processo que

instanciará o objeto *injector*. O campo plabel é o identificador do processo que instanciará o objeto *injector*. O campo glabel é o identificador do grupo que instanciará o objeto *injector*. O campo ilabel é o identificador do objeto *injector*. O formato em C++ é:

```
struct nameSetMessage
{
  header h;
 char messageType;
 char nodo[MAXLABELSIZE];
 char plabel[MAXLABELSIZE];
 char glabel[MAXLABELSIZE];
 char ilabel[MAXLABELSIZE];
};
```

b) Do objeto *controller-collector* para o *name_server* na criação dos objetos *injector*:

O campo messageType é setado para 20 para indicar que nameRequestMessage é uma mensagem de requisição de um nome ao servidor de nomes. O campo requestNumber indica o número de sequência de requisição. O campo nodo é o identificador do nodo onde se localiza o processo que instanciou o objeto *injector*. O campo plabel é o identificador do processo que instanciou o objeto *injector*. O campo glabel é o identificador do grupo que instanciou o objeto *injector*. O formato em C++ é:

```
struct NameRequestMessage
{
  header h;
 char messageType;
 int requestNumber;
 char nodo[MAXLABELSIZE];
 char plabel[MAXLABELSIZE];
 char glabel[MAXLABELSIZE];
};
```

c) Do objeto *name_server* para o *controller-collector* na criação de *injector*:

O campo messageType é setado para 30 para indicar que NameMessage é uma mensagem de resposta com a identificação do objeto *injector*. O campo requestNumber indica o número de sequência de requisição. O campo nodo é o identificador do nodo onde se localiza o processo que instanciou o objeto *injector*. O campo ilabel é o identificador do objeto *injector*. O formato em C++ é:

```
struct NameMessage
{
  header h;
 char messageType;
 int requestNumber;
 char ilabel[MAXLABELSIZE];
};
```

Bibliografia

[ARL 90] Arlat, J. et al. *Fault Injection for Dependability Validation: A Methodology and Some Applications*. IEEE Transactions on Software Engineering, v.16, n.2, p.166-182, Feb. 1990.

[BAR 90] Barton, J. H. et al. *Fault Injection Experiments Using FIAT*. IEEE Transactions on Computers, v.39, n.4, p.575-582, Apr. 1990.

[BAR 95] Barcelos, P. P. A; Weber, T. S. *Experimentação e avaliação de protocolos de difusão confiável*. In: Simpósio de Computadores Tolerantes a Falhas, VI SCTF. Canela 29 jul. a 4 aug. 1995. Anais. SBC. p. 277-290

[BOO 94] Booch, G. *Object-Oriented analysis and design with applications*. 2nd. Ed. The Benjamin/Cummings Publishing Company,Inc. 1994. 589p.

[BOO 96] Booch, G. *Object solutions: Managing the object-oriented project*. The Addison-Wesley Publishing Company,Inc. 1996. 323p.

[CAR 95a] Carreira, J. et al. *Xception: Software Fault Injection and Monitoring in Processor Functional Units*. DCCA' 95, Working Conference on Dependable Computing for Critical Applications, Urbana-Champaign, USA, Sep. 1995.

[CAR 95b] Carreira, J. et al. *Assessing the Effects of Communication Faults on Parallel Applications.* Proceedings of IPDS' 95, International Computer and Dependability Symposium, 1995, Erlangen, Germany, p.214-223.

[CHI 89] Chillarege, R.; Bowen, N. S. *Understanding large system failures - A fault injection experiment.* FTCS-19, 1989, p.356-363.

[CHO 92] Choi, G. S.; Iyer, R. K. *FOCUS: An Experimental Environment for Fault Sensivity Analysis.* IEEE Transactions on Computers, v.41, n.12, Dec. 1992.

[COU 94] Coulouris, G; Dollimore, J. and Kindberg, T. *Distributed Systems: Concepts and Design.* 2nd. Ed. The Addison-Wesley Publishing Company,Inc. 1994. 253-268p.

[DAW 94] Dawson, S.; Jahanian, F. *Probing and Fault Injection of Protocol Implementations.* Technical Report CSE-TR-217-94, The University of Michigan, 1994.

[DAW 95] Dawson, S. et al. *A Software Fault Injection Tool on Real-Time Mach.* Proceedings of IEEE Real-Time Systems Symposium. Dec. 1995.

[DAW 96] Dawson, S. et al. *Testing of Fault-Tolerant and Real-Time Distributed Systems via Protocol Fault Injection.* FTCS-26, Digest of Papers, IEEE Press, 1996.

[ECH 91] Echtle, K.; Chen, Y. *Evaluation of Deterministic Fault Injection for Fault-Tolerant Protocol Testing.* FTCS-21, Digest of Papers, IEEE Press, p.418-425, 1991.

[ECH 92] Echtle, K.; Leu, M. *The EFA Fault Injector for Fault-Tolerant Distributed System Testing.* Conf. Proc. Int. Symp. Fault-Tolerant Parallel and Distributed Systems, IEEE Press. p.28-35, 1992.

[ECH 94] Echtle, K. ; Leu, M. *Test of Fault Tolerant Distributed Systems by Fault Injection.* The 1994 IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems, conf. preprints, 1994.

[EZH 95] Ezhilchelvan, P. ; Macedo, R. A. and Shrivastava, S. *Newtop: A Fault-Tolerant Group Communication Protocol.* FTCS-25, Digest of Papers, IEEE Press, p.286-306, 1995.

[HAN 93] Han, S. et al. *DOCTOR: An IntegrateD SOftware Fault InjeCTiOn EnviRonment for Distributed Real-time Systems.* Technical Report CSE-TR-192-93, The University of Michigan, Dec. 1993.

[KAN 95] Kanawati, G. A. et al. *FERRARI: A Flexible Software-Based Fault and Error Injection System.* IEEE Transactions on Computers, v.44, n.2, p.248-260, Feb. 1995.

[KAO 93] Kao, W. et al. *FINE: A Fault Injection and Monitoring Environment for Tracing the UNIX System Behaviour under Faults.* IEEE Transactions on Software Engineering, v.19, n.11, p.1105-1118, Nov. 1993.

[KAR 94] Karlsson, J. et al. *Using Heavy-Ion Radiation to Validate Fault-Handling Mechanisms.* IEEE Micro, v.14, n.1, p.8-23, Feb. 1994.

[LAP 85] Laprie, J-C. *Dependable Computing and Fault Tolerance.* FTCS-15, 1985, p.2-11, Jun. 85.

[LOV 93] Lovric, T.; Echtle, K. *ProFI: Processor Fault Injection for Dependability Validation.* IEEE International Workshop on Fault and Error Injection for Dependability Validation of Computer Systems, Jun. 1993. Gothenburg. Sweden.

[MAR 93] Martins, E. *Validação de sistemas distribuidos tolerantes a falhas em presença de falhas.* V Simpósio de Computadores Tolerantes a Falhas , 1993, p.233-251.

[MAR 93a] Martins, E. *Validação Experimental da Tolerância a Falhas: A Técnica de Injeção de Falhas.* Mini Cursos do V Simpósio de Computadores Tolerantes a Falhas , 1993, p.56-70.

[ROS 93] Rosenberg, H. A. ; Shin, K. G. *Software Fault Injection and its Application in Distributed Systems.* FTCS-23, 1993, p.208-217.

[SEG 88] Segall, Z. et al. *FIAT - Fault Injection Based Automated Testing Environment.* FTCS-18, 1988, p.102-107.

[TEI 95] Teixeira Jr., C. A.; Weber, T. S. *FIX - uma ferramenta para recuperação de aplicações distribuidas no sistema operacional UNIX.* In: Simpósio de Computadores Tolerantes a Falhas, VI SCTF. Canela 29 jul. a 04 aug. 1995. Anais. SBC. p. 333-344.