MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

UMA ARQUITETURA DE TESTES PARA SISTEMAS ESPACIAIS

Cláudia Santos da Silva

Proposta de Dissertação de Mestrado em Computação Aplicada, orientada pelo Dr. Nandamudi Lankalapalli Vijaykumar e Dra. Eliane Martins

INPE
São José dos Campos
2005

MINISTÉRIO DA CIÊNCIA E TECNOLOGIA

INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-

UMA ARQUITETURA DE TESTES PARA SISTEMAS ESPACIAIS

Cláudia Santos da Silva

Proposta de Dissertação de Mestrado em Computação Aplicada, orientada pelo Dr. Nandamudi Lankalapalli Vijaykumar e Dra. Eliane Martins

“Democracia? É dar, a todos, o mesmo ponto de partida.
Quanto ao ponto de chegada, isso depende de cada um”.

MÁRIO QUINTANA

Dedico este trabalho às minhas filhas Ana Clara e Maria Fernanda, pelos momentos de alegria e descontração. E ao meu companheiro Renato que acreditou no meu esforço e contribuiu para esta realização. À vocês que mesmo sem entender direito, compreendeu minha ausência, quando os estudos me impediram de estarmos mais juntos.

"Somente a renúncia de tantos momentos, que poderiam ser vividos juntos, traduz a palavra amor".

Gilbran Khali

AGRADECIMENTOS

Agradeço à Deus pelo milagre da vida e pelas incontáveis bênçãos.

Aos meus pais, pelo amor e dedicação, sempre sacrificando seus sonhos em favor dos meus.

Às minhas irmãs pelo incentivo, compreensão e grande amizade.

Aos meus sogro e sogra, fonte de sabedoria e paciência, que a qualquer momento tem sempre um conhecimento a me transmitir.

Aos meus orientadores, Eliane e Vijay pela acolhida e imensa ajuda, amparando minhas dúvidas e suscitando em mim a maturidade profissional, sempre com muita paciência e dedicação.

Aos meus amigos, que me proporcionaram vários momentos de descontração mesmo em tempos de difícil compreensão.

RESUMO

Atualmente os computadores são utilizados em praticamente todas as áreas da ciência e tornaram imprescindíveis em várias atividades fundamentais na sociedade. Particularmente em aplicações espaciais, os sistemas computacionais têm que ser confiáveis e tolerantes à falhas, ou seja o sistema deve continuar operando de forma degradada ou não mesmo em presença de falhas. Para isso é necessário investir no processo de desenvolvimento de sistemas e fazer uso de alguns mecanismos de tolerância à falhas. Além disso, com a constante evolução tecnológica os sistemas para aplicações espaciais estão evoluindo e mudando de plataforma com uma freqüência cada vez maior. Isto faz com que seja necessário adaptar uma arquitetura para testes de software que suporte: testes multi-partes; várias tarefas sendo executadas em paralelo, característica inerente à sistemas espaciais; alta portabilidade; testes de injeção de falhas e que seja extensível. Assim esse trabalho apresenta uma arquitetura de testes para sistemas espaciais que contempla as características para esse tipo de sistema.

SUMÁRIO

Pág.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

AF	- Active Ferry
ATIFS	- Ambiente de Testes por Injeção de Falhas por Software
BPC	- Computador dos Experimentos Brasileiros
ComFIRM	- Communication Fault Injection through OS Resources Modification
Doctor	- Integrated Software Fault Injection Environment
ETBPC	- Equipamento de Testes do Computador dos Experimentos Brasileiros
FBM	- Microssatélite Franco-Brasileiro
FIC	- Fault Injection Controller
FIESTA	- Fault Injection for Embedded System Target Application
FIM	- Fault Injection Mobile
FSM	- Finite State Machine
FSoFIST	- Ferry-clip with Software Fault-Injection Support Tool
FSoFIST-mp	- Ferry-clip with Software Fault-Injection Support Tool with multi parts
INPE	- Instituto Nacional de Pesquisas Espaciais
ISO	- Interconnection Software Open
IUT	- Implementation Under Test
LMAP	- Lower Mapping Module
MFEE	- Máquina Finita de Estado Estendida
OBC	- On Board Computer
OMT	- Técnica de Modelagem de Objetos
OOSE	- Object Oriented Software Engineering
OSI	- Open Systems Interconnection
PF	- Passive Ferry

PFI	- Protocol Fault Injection – Injeção de Falhas de Protocolo
SAI	- Service Interface Adapter
SDT	- Sistema de Desenvolvimento de Testes
SGS	- Subsistema de Geração de Scripts
SOAT	- Software de Análise de Telemetria
SOCA	- Software de Controle da Placa AD/DA
SOGA	- Software para Geração de Pacotesde TM dos Experimentos
SSE	- Subsistema de Suporte à Execução
STR	- Subsistema de Tratamento dos Resultados
TC	- Tele Comando
TM	- TeleMetria
UML	- Unified Modelling Language
Xception	- Software Fault Injection and Monitoring in Processor Functional Units
XML	- eXtensible Markup Language

CAPÍTULO 1

INTRODUÇÃO

1.1. Contexto

Atualmente os computadores são utilizados em praticamente todas as áreas da ciência e tornaram imprescindíveis em várias atividades fundamentais na sociedade. Em alguns casos são vidas humanas que dependem deles, como sistemas médicos, controle de tráfego aéreo, controle de trajetória de foguetes entre outros, e estas aplicações são consideradas críticas, porque qualquer tipo de falha pode ter conseqüências graves não só em termos financeiros, mas até em vidas humanas.

Em aplicações espaciais a principal característica dos sistemas computacionais é a confiabilidade, ou seja, o sistema deve ter uma alta probabilidade de funcionar de acordo com o especificado durante um intervalo de tempo pré-definido.

Como não é possível prevenir completamente a ocorrência de falhas que possam provocar o mau funcionamento do sistema, torna-se imprescindível também o uso de alguns mecanismos de tolerância a falhas, que façam com que o sistema continue operando (de forma degradada ou não) mesmo após a ocorrência de falhas no sistema. Além disso, é necessário investir no processo de desenvolvimento do sistema (especificação, projeto, implementação e testes), ou seja, no processo de prevenção de falhas.

Vale a pena ressaltar que as falhas de sistemas computacionais em aplicações espaciais podem provocar a perda da missão e conseqüentemente provocar grandes perdas econômicas (ou até risco às vidas humanas) e, assim sendo eles são classificados como sistemas críticos relacionados ao aspecto financeiro.

Assim no contexto de desenvolvimento de sistemas o *Teste de Software* torna-se elemento fundamental da garantia da qualidade que representa a última etapa de revisão da especificação, do projeto e do código. O teste é uma forma de verificação que consiste em executar o programa com um conjunto de dados de entrada e determinar se ele se comporta conforme o esperado [Pressman 1997].

Além da verificação, o software necessita ser validado e uma das formas de validar o projeto/ implementação de sistemas tolerantes a falhas é através de ferramentas de injeção de falhas. Estas ferramentas injetam falhas durante a execução dos testes e fornecem dados sobre o comportamento do sistema na presença destas falhas injetadas. A injeção de falhas implementada por software tem sido cada vez mais usada para validar sistemas críticos.

1.2. Motivação

A constante evolução tecnológica tem feito com que sistemas em aplicações espaciais evoluam e mudem de plataformas com uma freqüência cada vez maior, tornando necessário a criação ou adaptação de uma arquitetura para teste de software apoiada ao uso de ferramentas para execução de testes e injeção de falhas no sistema.

Atualmente o INPE (Instituto Nacional de Pesquisas Espaciais) não conta com nenhuma arquitetura de testes e nem uma ferramenta que possa apoiar os testes para os sistemas desenvolvidos. Em aplicações espaciais os testes mais comuns são os testes de conformidade, que visam determinar se uma determinada implementação está de acordo com a especificação do sistema e os testes de desempenho que avaliam se a implementação do sistema apresenta o desempenho esperado.

O Instituto da Computação da Unicamp e o Instituto Nacional de Pesquisas Espaciais mantêm o ATIFS *(Ambiente de Testes de Injeção de Falhas por Software)* [Martins 1995], um projeto cujo objetivo é dar suporte à estratégia de teste por injeção de falhas e a outros tipos de testes, tais como testes de conformidade. Esse projeto viabiliza o uso de várias ferramentas envolvidas no processo de testes.

1.3. Objetivo

O objetivo deste trabalho é adaptar uma arquitetura de testes de software para aplicações espaciais com suporte a testes multi-partes concentrando nos testes de conformidade e testes de injeção de falhas. Para isso será utilizada a arquitetura Ferry Injection [Araújo 2000] como base para a adaptação da arquitetura Ferry Injection-mp (Ferry-Injection Multi-Partes) que é a arquitetura proposta neste trabalho.

Visando este objetivo, a ferramenta FSoFIST-mp *(Ferry-clip with Software Fault-Injection Suppot Toll Multi-Partes)*, foi projetada e parte desta ferramenta foi desenvolvida baseando-se na arquitetura Ferry-Inection-mp para executar testes de conformidade e testes de injeção de falhas.

1.4. Contribuição do trabalho

A contribuição principal deste trabalho é dar continuidade ao projeto ATIFS e oferecer ao INPE uma arquitetura que apóie os testes de conformidade e testes de injeção de falhas por software para aplicações espaciais, em especial para sistemas embarcados.

1.5. Estrutura da dissertação

Essa dissertação esta estruturada da seguinte forma:

O capítulo 2 apresenta uma das bases teóricas deste trabalho, que são os testes de conformidade de sistemas de comunicação e a arquitetura ferry clip que foi desenvolvida para suportar as metodologias de testes para testes de conformidade. O capítulo 3 apresenta o projeto ATIFS, dá uma visão geral dos conceitos sobre injeção de falhas e apresenta diferentes trabalhos realizados nesta área. O capítulo 4 expõe a arquitetura para sistemas espaciais proposta e os requisitos de uma ferramenta visando facilitar a realização de testes em aplicações espaciais. O capítulo 5 apresenta uma análise da arquitetura baseando-se num sistema real do INPE. O capítulo 6 apresenta as conclusões e sugere possíveis trabalhos futuros que poderiam ser feitos com base na arquitetura proposta.

CAPÍTULO 2

TESTES DE CONFORMIDADE E ARQUITETURA DE TESTES

2.1. Introdução

Este Capítulo apresenta um resumo de um dos principais conceitos relacionados a testes de sistemas, a arquitetura de testes, enfocando mais especificamente nas arquiteturas para testes de conformidade. A Seção 2.2 apresenta uma definição de testes de conformidade e um resumo do padrão IS9646 definido pela ISO *(Interconnection Software Open)* apresentando as arquiteturas definidas por esse padrão. Na Seção 2.3 é apresentada a arquitetura Ferry Clip [Chanson 1992] que foi desenvolvida para dar suporte às metodologias definidas pela ISO. A Seção 2.4 apresenta a arquitetura ferry injection [Araújo 200] que foi criada baseada na arquitetura ferry clip e que foi escolhida como base à arquitetura proposta. A Seção 2.5 apresenta um estudo realizado sobre arquiteturas multi-partes baseadas na ferry clip, e uma análise do porquê nenhuma delas contemplou os requisitos definidos.

2.2. Testes de Conformidade

Testes de conformidade têm por objetivo determinar se uma dada implementação satisfaz os requisitos definidos em sua especificação. No contexto do Modelo de Interconexão para Sistemas Abertos (ou OSI, de *Open Systems Interconection*), foi desenvolvido um padrão para os testes de conformidade do modelo OSI. É o padrão IS9646: "*OSI Conformance Testing Methodology and Framework*" [ISO]. O padrão define a metodologia, a estruturação e especificação de seqüências de testes, além de procedimentos a serem seguidos durante os testes. O intuito é permitir que os resultados de testes realizados por diferentes grupos sejam comparáveis e reprodutíveis, evitando assim a duplicação de esforços. O padrão não estabelece como os testes devem ser gerados, mas define um arcabouço para a estruturação dos testes, bem como a forma de especificá-los. O padrão define também as arquiteturas de apoio à execução dos testes.

26

Segundo o padrão as arquiteturas (também chamadas de métodos) de testes devem basear-se numa metodologia abstrata de testes, cujo principal objetivo é a especificação dos Pontos de Controle e Observação (PCO) das entradas e saídas da implementação sob testes (IUT). A arquitetura conceitual de testes pode ser descrita em termos de [Tretmans 1999]:, (i) acessibilidade aos PCOs, (ii) contexto de teste, que é o ambiente que a IUT está embutida durante os testes, (iii) testadores, que são associados aos PCOs, e são chamados de Testador Superior (TS) e Testador Inferior (TI), conforme mostrado na figura 2.1.

Existem casos nas quais a implementação sob testes e os testadores podem residir no mesmo ambiente físico e outros em que a IUT reside em um ambiente remoto, para isso foram propostas diversas arquiteturas: local, distribuída e remota.

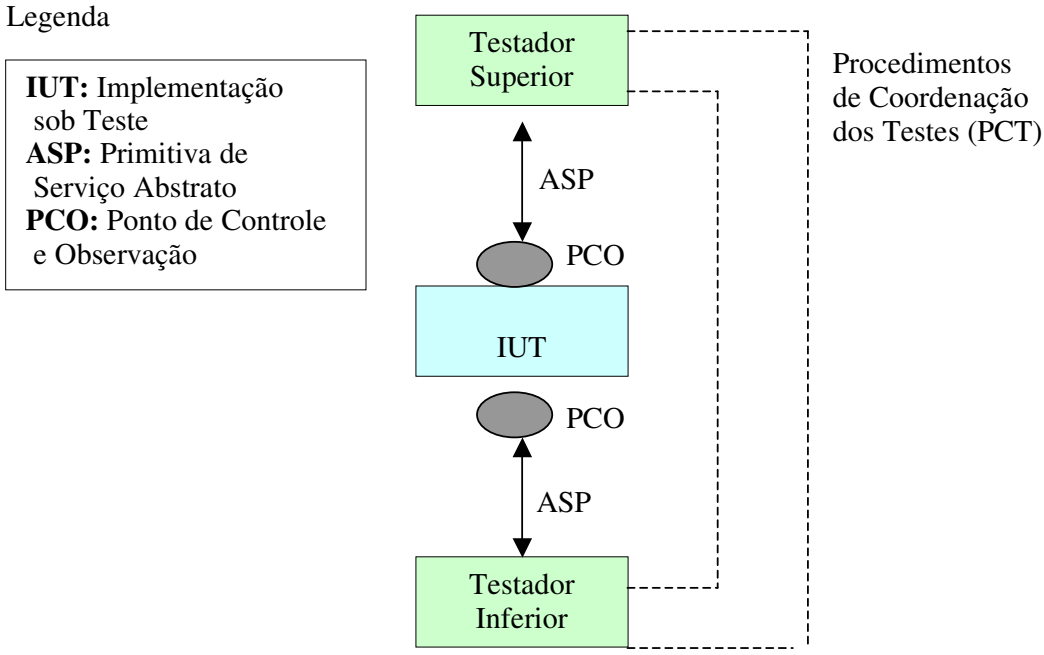


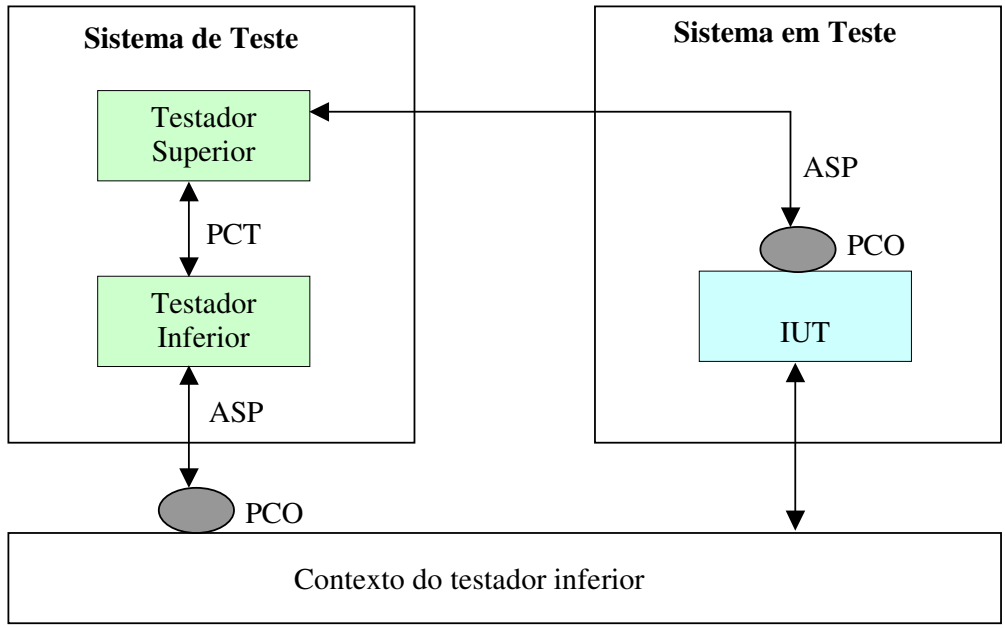
FIGURA 2.1. Arquitetura conceitual para os testes de conformidades

27

2.2.1 Arquitetura de teste local

Na arquitetura de teste local a IUT interage tanto com o Testador Superior (TS) quanto com o Testador Inferior (TI). Esta arquitetura, mostrada na Figura 2.2, intercepta as Primitivas de Serviço Abstrato (ASP) nas interfaces superior e inferior da IUT, e estas interfaces constituem os Pontos de Controle e Observação (PCO) usados para os testes.

As arquiteturas de testes externos se caracterizam pelo fato de que o TI só tem acesso remotamente à interface inferior da IUT. O procedimento de Coordenação dos Testes (PCT) deve implementar um protocolo para a comunicação entre o TS e o TI.



Legenda

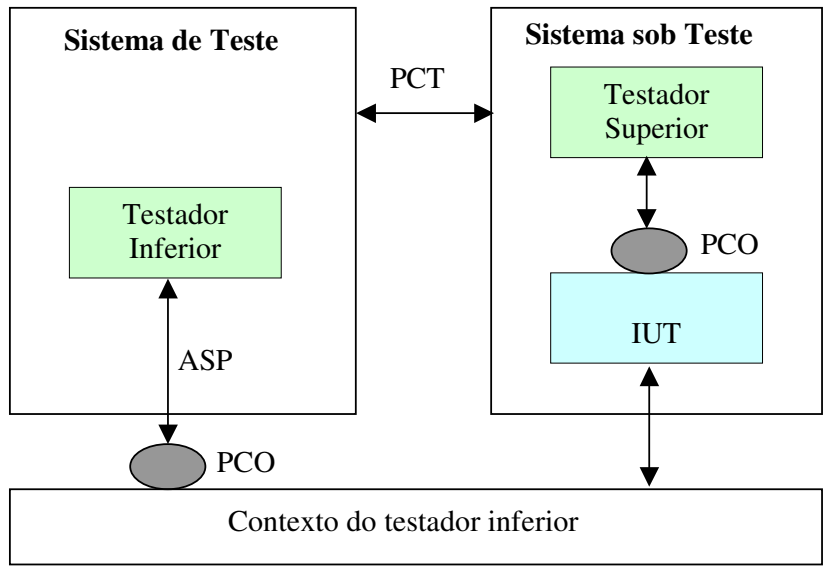
IUT: Implementação sob Teste
ASP: Primitiva de Serviço Abstrato
PCO: Ponto de Controle e Observação
PCT: Procedimentos de Coordenação dos Testes

FIGURA 2.2. Arquitetura de teste local

28

2.2.2. Arquitetura de teste distribuído

Esta arquitetura é parecida com a arquitetura local para aplicações de testes no que se refere à interceptação das ASPs. A diferença está no fato do Testador Superior localizar-se junto a IUT no sistema em teste, como mostra a Figura 2.3. Neste caso os procedimentos de teste são distribuídos entre o Sistema de Teste, que fica responsável pela interface inferior e o Sistema em Teste com a interface superior. A coordenação entre os testadores fica sob responsabilidade do usuário.



Legenda

IUT: Implementação sob Teste
ASP: Primitiva de Serviço Abstrato
PCO: Ponto de Controle e Observação
PCT: Procedimentos de Coordenação dos Testes

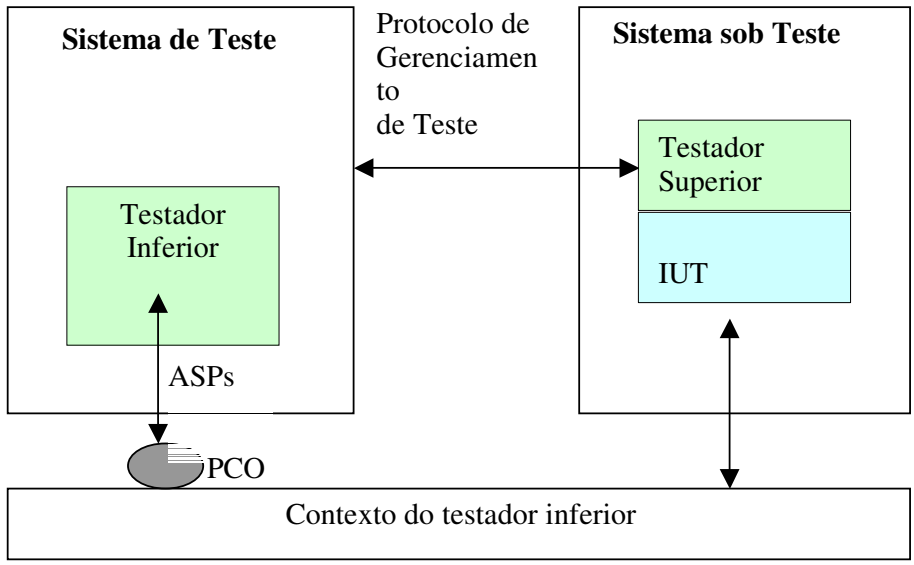
FIGURA 2.3. Arquitetura de teste distribuído

2.2.3. Arquitetura de teste coordenado

Na arquitetura de teste coordenado a aplicação dos testes é feita diretamente na interface superior da Implementação sob Testes. Neste caso a coordenação dos testes é

29

feita através de um protocolo para troca de dados entre as partes, como mostra a Figura 2.4.



Legenda

IUT: Implementação sob Teste **ASP:** Primitiva de Serviço Abstrato **PCO:** Ponto de Controle e Observação

FIGURA 2.4. Arquitetura de teste coordenado.

2.2.4. Arquitetura de teste remoto

Nesta arquitetura o testador superior é suprimido e os testes são realizados apenas através da interface inferior. É utilizado quando não se tem acesso à interface superior da implementação em teste. A Figura 2.5 mostra esta arquitetura.

30

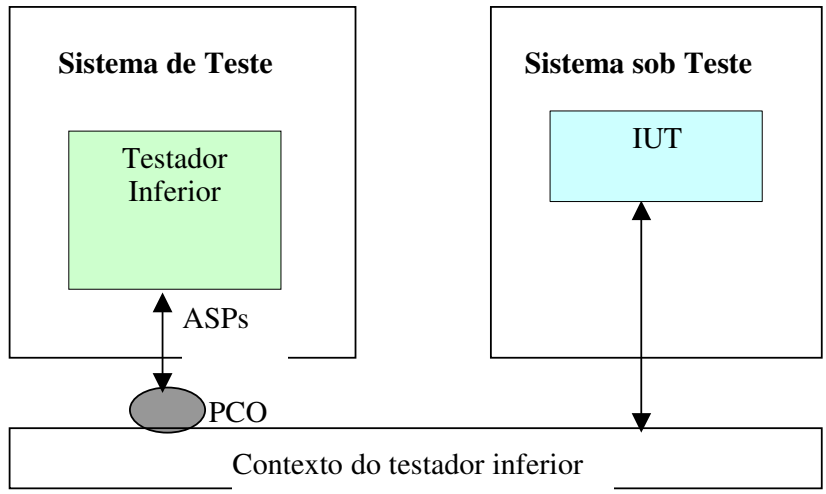


FIGURA 2.5. Arquitetura de teste remoto.

2.3. Arquitetura Ferry Clip

A arquitetura *Ferry Clip* [Chanson] foi desenvolvida com o objetivo de dar suporte às metodologias de testes definidas pela ISO. A idéia básica é permitir que os dados sejam trocados entre o Sistema em Teste (Sistema alvo) e o Sistema de Teste de maneira transparente permitindo que tanto o Testador Superior quanto o Testador Inferior residam junto ao sistema de teste, simplificando a sincronização entre os testadores e minimizando a quantidade de software residente no sistema em teste.

Os principais componentes da arquitetura *ferry* são dois *ferry-clips*, chamados de *Ferry* Ativo (*Active Ferry* – AF) e o *Ferry* Passivo (*Passive Ferry* – PF) que trocam dados entre si através de um protocolo de comunicação denominado Canal do *Ferry*. A Figura 2.6 apresenta a arquitetura *ferry-clip*.

O sistema de teste é formado pelo AF e pela Máquina de Testes *(Test Engine),* que agrupa as funções do Testador Superior e do Testador Inferior, sendo também responsável pelo envio dos comandos à máquina de estados finita do AF. Estes comandos servem para estabelecer conexão com o PF, pedidos de desconexão e outros dados de controle específicos. O sistema em teste é formado pelo PF e pela IUT. As informações são trocadas entre o AF e o PF através de primitivas de serviço abstrato, que estão descritas a seguir:

31

- *Ferry Data* (FD-asp): primitiva de transporte de dados entre a Máquina de Teste e a implementação em teste;
- *Ferry Management* (FM-asp): para envio de comandos às FSM's;
- *Ferry Transport* (FT-asp), para a troca de dados entre o AF e o PF.
- A máquina de estados finita do AF e a do PF trocam dados através de pacotes de dados FY-PDU's *(Ferry Protocol Data Units),* encapsuladas pelo *Ferry Transport* (FT-asp).

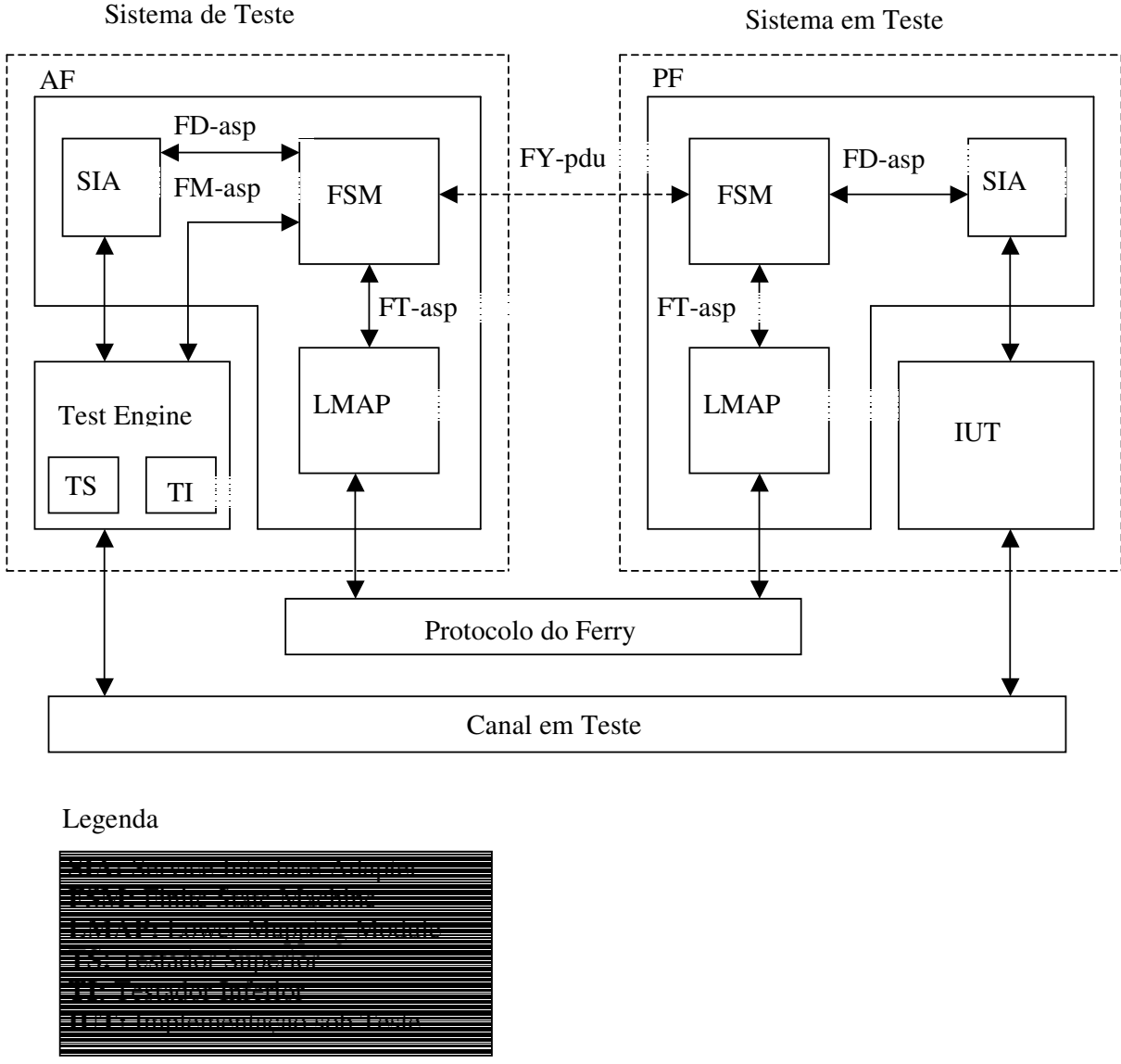


FIGURA 2.6. Arquitetura Ferry Clip

32

2.3.1. AF (Active Ferry)

O Sistema de Teste é formado pelo *Test Engine*, onde são agrupadas as funções do Testador Superior e do Testador Inferior e pelo AF. O AF é responsável por iniciar a conexão com o PF e por transferir os dados dos testadores para o sistema em teste, e é implementado independente da implementação em teste. Assim uma vez que a IUT é trocada, o AF não é afetado, permitindo a sua reutilização em outras aplicações. O AF é dividido em três partes:

- *Service Interface Adapter* (SIA): este componente é responsável por adaptar a implementação em teste para que esta fique disponível para o sistema de teste de maneira que haja o transporte das interfaces. Caso uma implementação em teste seja substituída este é o único módulo afetado;
- *Finite State Machine* (FSM): incorpora todas as funções independentes do protocolo do ferry. Ela é responsável pelo controle da comunicação entre o AF e o PF, isto é, solicitação de conexão e desconexão, envio e recepção de dados;
- *Lower Mapping Module* (LMAP): responsável pela troca de dados entre o AF e o PF através do protocolo do ferry. Caso o protocolo seja trocado este é o único módulo afetado no AF.

2.3.2. PF (Passive Ferry)

O PF reside no sistema em teste e também possui código necessário para estabelecer e manter conexão com o AF. A responsabilidade do PF é o encapsulamento e envio dos dados provenientes da implementação em teste para o sistema de teste através do AF e de aplicar os testes provenientes do sistema de teste na IUT. O módulo PF é também dividido em três partes:

- *Service interface Adapter:* transporta a interface da implementação em teste até o sistema de teste através do AF, recebendo dados desta e convertendo-os em um formato compreensível ao sistema de teste. No caso de alteração da IUT, este é o único módulo que é alterado;

33

- *Finite State Machine:* encerra todas as funções independente do protocolo do ferry. Este módulo é responsável pelo recebimento e processamento das solicitações feitas pelo AF, bem como as solicitações feitas pela IUT;
- *Lower Mapping machine:* encerra todas as funcionalidades para a troca de dados entre AF e PF através do protocolo do ferry. Caso o protocolo do ferry seja trocado este é o único módulo afetado no PF.

Esta arquitetura Ferry Clip foi escolhida como base à adaptação da arquitetura Ferry-Injection-mp proposta, porque apresenta as seguintes facilidades: de portar para diferentes plataformas; de adaptar a diferentes implementações; no acréscimo de novas funcionalidades, ou seja, expansão da arquitetura com mínima interferência. Não é necessário desenvolver uma nova ferramenta para cada implementação em teste desde que se modifique apenas o adaptador de interface de serviço em cada um dos módulos, deixando-os com interface disponível à máquina de testes. Do mesmo modo, basta modificar os módulos do LMAP para diferentes canais de conexão entre o AF e PF. Assim pode-se aplicar os mesmos casos de testes em IUTs diferentes sem que haja necessidade de grandes modificações na ferramenta.

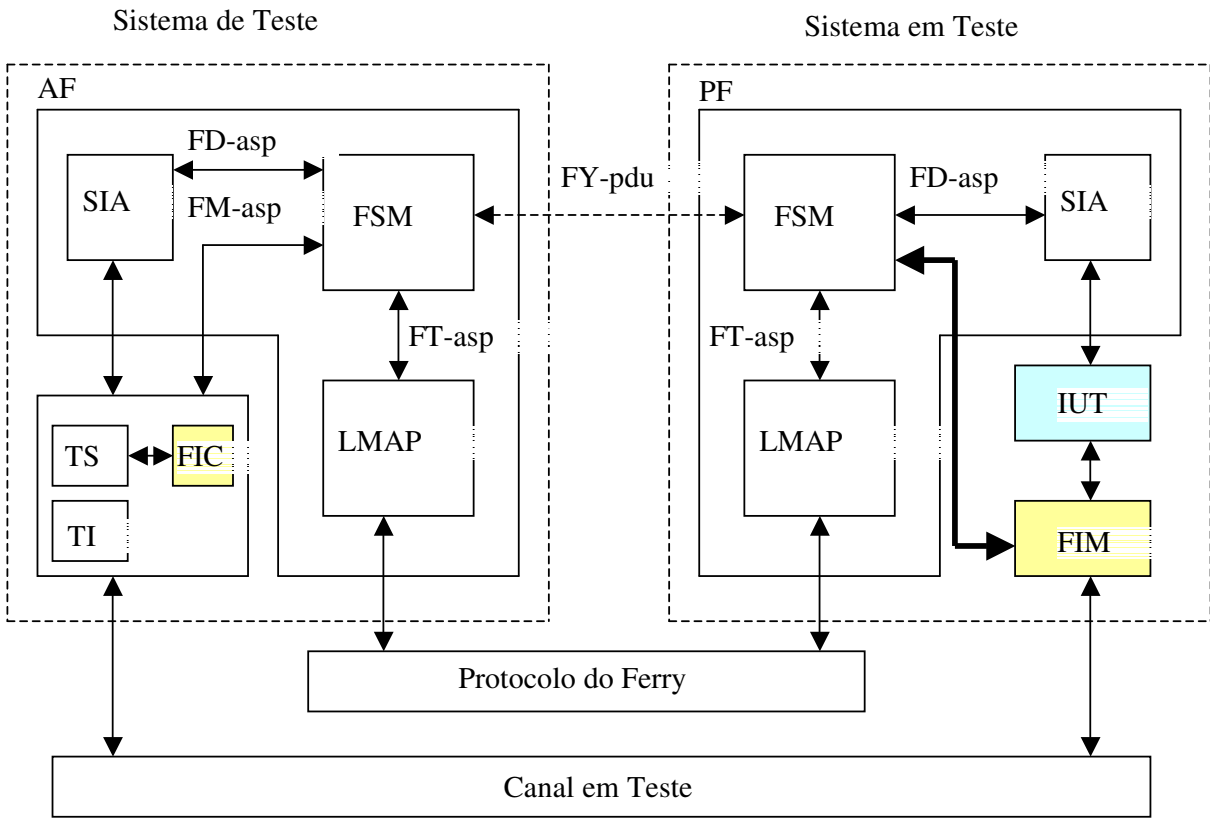
2.4. Arquitetura Ferry Injection

A arquitetura ferry-Inection [Araújo 2000] também é baseada na arquitetura ferry clip, na qual a idéia é permitir a injeção de falhas. Esta extensão consistiu em acrescentar um injetor de falhas junto à IUT e um mecanismo de controle deste injetor junto ao sistema de teste. O mecanismo de troca de mensagens é o mesmo utilizado para enviar dados à IUT. A Figura 2.7 apresenta a arquitetura ferry injection.

Comparando as arquiteturas ferry clip e ferry injection pode-se verificar as seguintes alterações:

- Criação de um canal lógico de transporte entre a máquina de testes e o módulo injetor de falhas.
- Um submódulo FIM acrescentado ao sistema em teste desempenhando a função de injeção de falhas.

- Um submódulo FIC acrescentado à máquina de testes exercendo a função de controlar as falhas injetadas.



SIA: Service Interface Adapter
FSM: Finite State Machine
LMAP: Lower Mapping Module

FIGURA 2.7. Arquitetura Ferry injection

2.5. Arquiteturas Multi-Partes

Esta seção apresenta um estudo das configurações da arquitetura *ferry* para testes multi-partes, conforme descrito em [Fischer 1989].

A Figura 2.8 apresenta um exemplo de arquitetura distribuída multi-partes. Um dos nós representa a IUT e os outros nós representam implementações denotadas por

35

CI^1 a CI^5. Nesta configuração o nó CI^4 é utilizado como IUT. Em função da própria natureza dos sistemas distribuídos multi-partes a configuração e análise dos resultados dos testes é dificultada pela restrição existente na observação do comportamento da IUT e dos nós vizinhos durante a comunicação. Além disso, esta configuração não é bem aplicada aos testes de conformidade pela falta de funcionalidade em expor a IUT a comportamentos incorretos entre os pares.

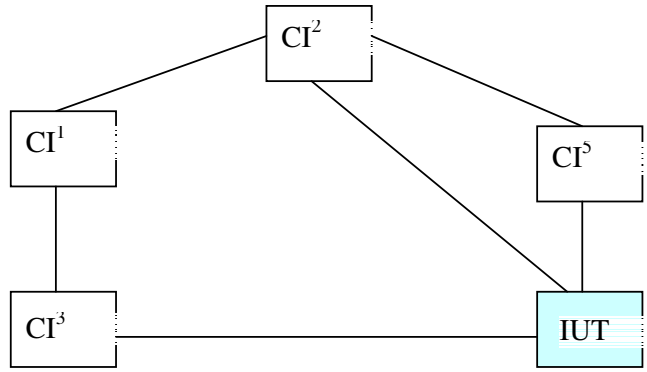


FIGURA 2.8. Teste multi-partes sem equipamento de teste

A Figura 2.9 [Fischer 1989] apresenta outra configuração desse tipo de sistema, porém introduzindo um sistema de teste, no qual cada nó da rede é conectado a este sistema via canal do *ferry,* possibilitando assim também a realização de testes de interoperabilidade. Cada componente pode ser seletivamente controlado por meio de Primitivas de Serviço Abstrato transportadas via o canal do *ferry*. Nesta configuração o sistema de teste tem controle das atividades de comunicação na rede, podendo observar o comportamento da IUT e das implementações dos nós que estão diretamente ligadas a ela. Com isso torna-se mais viável a utilização dessa configuração do que a mostrada na Figura 2.8 para testes de conformidade, porém há também a dificuldade ou até mesmo, impossibilidade dentro deste ambiente de expor a IUT a um comportamento inválido dos pares.

36

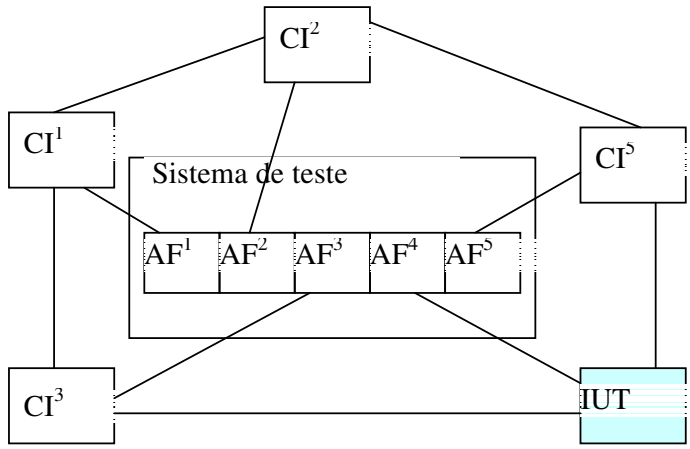


FIGURA 2.9. Teste multi-partes com sistema de teste utilizando ferry

Uma outra configuração estudada em [Fischer 1989] é apresentada na Figura 2.10. Esta configuração mostra a extensão de uma rede virtual modelada dentro do sistema de teste. Isto é realizado simulando a comunicação entre as implementações dos nós no sistema de teste, no qual somente as implementações que são visíveis a IUT são requeridas. No exemplo de rede considerado, as implementações necessárias são a CI^2, CI^3 e CI^5, que mantém três canais de teste com a IUT. A presença do CI^1 no exemplo de rede é simulado somente se este influenciar o comportamento das demais implementações.

Nesta configuração o sistema de teste exerce controle total do comportamento da comunicação com a IUT na rede, expondo a IUT ao possível comportamento inválido dos pares.

A vantagem dessa configuração é que pode-se mudar a topologia de testes sem qualquer modificação no ambiente de teste real, uma vez que a diferença entre as topologias visíveis pela IUT é o número de canais de teste.

Mesmo com as vantagens desta configuração, difere muito da arquitetura existente no ATIFS. No projeto ATIFS o protótipo da ferramenta FSoFIST (*Ferry-clip with Software Fault-Injection Support Tool)*, a IUT comunica-se com o sistema de teste somente via o AF, não existindo outra forma de comunicação das implementações no

37

sistema de teste com a IUT. Como a proposta é adaptar a arquitetura já existente no projeto ATIFS para testes multi-partes, esta arquitetura não será adotada.

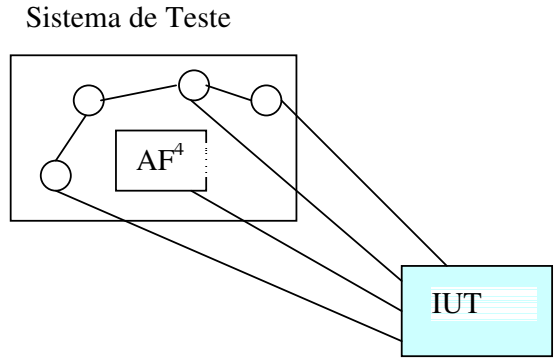


FIGURA 2.10. Cofiguração multi-partes para testes de conformidade

A Figura 2.11 apresenta uma configuração da arquitetura multi-partes proposta em [Chanson 1992]. O estudo desta configuração mostra como a arquitetura *ferry clip* pode ser configurada para encontrar os requisitos dentro do escopo da OSI. Porém esta configuração também não se aplica ao estudo que está sendo realizado, pois na arquitetura do ATIFS a IUT comunica-se com o sistema de teste apenas via canal do *ferry*, não existindo, portanto, um testador inferior comunicando-se diretamente com a IUT como acontece na arquitetura apresentada abaixo.

38

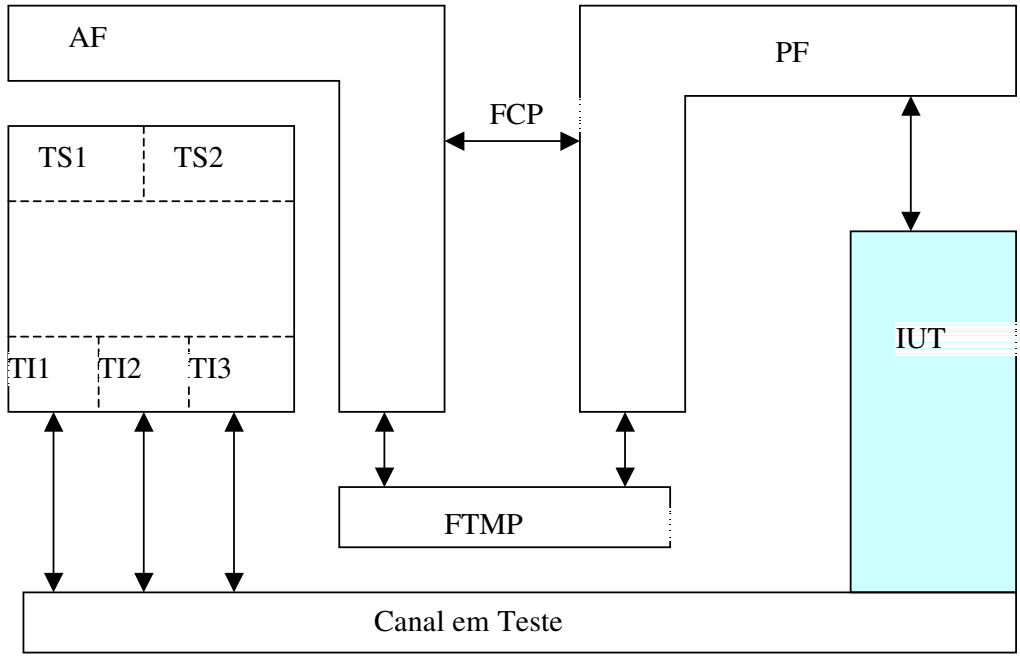


FIGURA 2.11. Configuração multi-partes baseado na ferry-clip para camada de transferência de mensagens

39

CAPÍTULO 3

INJEÇÃO DE FALHAS

3.1. Introdução

Este Capítulo apresenta o conceito de injeção de falhas, os tipos de injeção de falhas e o estudo feito de algumas ferramentas desenvolvidas nesta área. Este estudo tem como objetivo servir de base para a definição das principais características e requisitos de uma ferramenta que poderá ser utilizada na arquitetura proposta. A Seção 3.2 apresenta uma das definições dadas à injeção de falhas, suas técnicas e alguns métodos. Na Seção 3.3 é apresentado um resumo de algumas ferramentas de injeção de falhas, incluindo a ferramenta proposta e uma comparação entre elas. A Seção 3.4 apresenta uma arquitetura genérica para Injeção de Falhas formulada [Hsueh 1997] a partir de estudos sobre ferramentas de injeção de falhas. E a Seção 3.5 apresenta o projeto ATIFS.

3.2. Injeção de Falhas

A importância de projetar e implementar sistemas confiáveis têm crescido e, conseqüentemente, também cresceu a importância de testar esses sistemas. Assim, algumas técnicas de teste para aumentar a confiabilidade de sistemas foram desenvolvidas, e entre elas a mais popular é a Injeção de falhas.

A injeção de falhas consiste na introdução de falhas ou erros em um sistema com o objetivo de observar o seu comportamento. Essa característica faz com que esta seja uma técnica muito útil para validação de mecanismos de recuperação de erros ou ainda para determinar se um sistema apresenta comportamento de risco em presença de falhas.

Geralmente a técnica de injeção de falhas é utilizada em sistemas ou componentes tolerantes as falhas, ou seja, o sistema deve continuar operando mesmo na presença de falhas e deve apresentar segurança no funcionamento (dependability).

Neste ponto, é necessário dar uma melhor definição de falha. Um defeito do sistema ocorre quando o serviço requerido foi desviado de sua especificação. Um erro é o estado do

40

sistema que levou ou pode levar a um defeito. E finalmente, a causa suposta ou real de um defeito, é a falha.

3.2.1. Técnicas de Injeção de Falhas

A injeção de falhas pode ser feita de três maneiras:

- Injeção de falhas por simulação: Esta técnica é usada na fase de projeto e as falhas são introduzidas num modelo do sistema alvo. Esta técnica é muito útil para avaliar dependabilidade do sistema nas primeiras fases do desenvolvimento e tem a vantagem de proporcionar grande controle e observação das falhas injetadas quando comparado com injeção num protótipo ou sistema final. A desvantagem é o alto custo de desenvolvimento devido à necessidade de se representar o sistema em teste através do software e não ter a garantia de que a simulação corresponderá à implementação real;
- Injeção de falhas por hardware: Nesta técnica as falhas são originadas de algum hardware especial, projetado especialmente para esse propósito. Existem várias técnicas para injetar falhas de hardware: alteração de níveis lógicos em pinos de circuitos integrados [Madeira 1991], por irradiação de íons pesados [Gunnelo 1989] entre outros. Esta técnica é utilizada para verificar a eficácia de mecanismos tolerantes a falhas implementadas em hardware. Contudo, tem um alto custo de implementação devido à necessidade de desenvolvimento de um hardware próprio para esta finalidade, além da possibilidade de causar danos à implementação.
- Injeção de falhas por software: Esta técnica procura injetar falhas no nível de software: sistemas operacionais e aplicações [Hsueh 1997]. Não é preciso hardware especial, e os testes podem ser facilmente controlados. O foco desta técnica é alterar o estado do sistema alvo, sob controle de um software, emulando dessa forma tanto as conseqüências de falhas de hardware quanto de software. Consiste em interromper a execução do sistema alvo através de algum mecanismo e executar o código de um injetor de falhas. Por causa dessas

41

vantagens, injeção de falhas por software tem ficado mais popular entre os desenvolvedores de sistemas tolerantes a falhas.

Outro ponto importante na injeção de falhas são os métodos utilizados para injetá-las. Uma maneira de categorizar essas formas é definir quando elas são injetadas, se em tempo de compilação ou em tempo de execução [Hsueh 1997]. A seguir são apresentados os métodos mais comuns de injeção de falhas por software, que é o método mais adequado para este estudo.

3.2.2. Métodos de Injeção de Falhas

Na injeção em *tempo de compilação*, as falhas são introduzidas no código fonte ou *assembly* do programa alvo, e consiste na alteração de instruções do programa. Na execução, quando uma instrução modificada é executada, a falha é ativada. Este método não precisa do software extra durante a execução para injetar falhas. Como resultado, não causa perturbação no sistema alvo durante a execução.

Em injetores em tempo de execução, código extra é necessário para injetar falhas e monitorar seus efeitos, um injetor de falhas e um monitor, respectivamente. Além disso, também é preciso um mecanismo para disparar a injeção de falhas.

Os métodos mais comuns de injeção de falhas são:

- *Time-out*, neste método um temporizador de hardware ou software é utilizado para gerar os eventos de *time-out* e a injeção é disparada a cada *time-out*. Este método não necessita de modificação no programa alvo: a injeção de falhas pode ser a monitoração e pode ser implementada como parte de uma rotina de interrupção.
- *Traps*, também é baseada na interrupção causada por hardware ou software, mas numa maneira diferente porque as falhas são disparadas por eventos que não sejam time-outs.
- Modo Traço, disponível em microcomputadores ou depuradores. Neste método, o programa é executado passo a passo, e uma rotina de traço é executada antes de cada instrução do programa alvo.

42

- Inserção de código, consiste em adicionar código ao programa alvo que permita que a injeção de falhas ocorra antes da execução de uma instrução específica. Diferente dos métodos acima, o código de injeção de falhas é parte do programa alvo e roda em modo usuário, e não em modo supervisor ou com algum privilégio.

A escolha do método mais apropriado depende de vários fatores, entre elas: o nível aceitável de perturbação do programa alvo, acessibilidade dos componentes nos quais as falhas serão injetadas, recursos disponíveis pelo ambiente de execução e objetivo dos experimentos. Assim, as ferramentas de injeção de falhas estão combinando vários métodos, e a próxima seção mostra algumas ferramentas onde os vários métodos combinados.

3.3. Ferramentas de Injeção de Falhas

Muitas ferramentas de injeção de falhas por software foram desenvolvidas, utilizando vários métodos para injetar falhas, a seguir são apresentadas algumas dessas ferramentas.

3.3.1. ORCHESTRA

ORCHESTRA [Dawson] é uma ferramenta que utiliza injeção de falhas para testar implementações de protocolos em sistemas distribuídos. Foi desenvolvido pela Universidade de Michigan. Esta ferramenta considera um protocolo como sendo uma pilha de diferentes camadas. As mensagens trocadas entre nós num sistema distribuído são geradas e passadas para a camada mais abaixo, que por sua vez passa para camada imediatamente mais abaixo, e assim sucessivamente até chegar na camada mais baixa e assim transportada para o nó destino. No nó destino, as mensagens chegam na camada mais baixa e o processo é revertido, até que cheguem na camada mais acima e são passadas para a aplicação. A ferramenta ORCHESTRA insere uma nova camada numa pilha já existente, chamada camada PFI (*Protocol Fault Injection* – Injeção de Falhas de Protocolo). O objetivo é modificar, apagar ou adicionar mensagens e assim injetar falhas e testar se o protocolo distribuído pode lidar com essas falhas.

43

As vantagens desta ferramenta são: não causa quase nenhuma perturbação, uma vez que as camadas são independentes; facilidade para adaptar para testar outros protocolos diferentes. A principal desvantagem é que pode ser usada apenas para sistemas baseados em protocolos, sendo assim pode somente injetar falhas relacionadas a transmissão de mensagens por sistemas distribuídos.

3.3.2. Xception

Xception *(Software Fault Injection and Monitoring in Processor Functional Units)* [Carreira 1995] foi desenvolvida pela Universidade de Coimbra, Portugal, utiliza processadores modernos com características especiais. Esses processadores têm algumas instruções especiais com o objetivo de realizar depuração e testes de performance dos chips. Esta ferramenta é utilizada para injetar falhas de hardware, onde essas falhas são disparadas pela ocorrência de algum evento específico, causando uma exceção no hardware. Como exemplo, considere um processador quando atinge uma dada instrução no código do sistema. Ocorrendo uma exceção, a execução pode ser desviada para um ponto do código escolhido aleatoriamente, simulando uma falha do ponteiro de instrução ou de barramento de código.

A vantagem dessa ferramenta também é a não perturbação no sistema sob teste, uma vez que não é necessário código para fazer a injeção de falhas e a monitoração. As desvantagens dessa ferramenta são: a dependência das instruções especiais de depuração dos processadores, pois cada vez que modificar o processador a ferramenta tem que ser reescrita; outra desvantagem é a disponibilidade do uso dessas instruções que depende de um acordo com os fabricantes do chip.

3.3.3. ComFIRM

ComFIRM *(Communication Fault Injection through OS Resources Modification)* [Barcelos 1999], foi desenvolvida na Universidade Federal do Rio Grande do Sul, Brasil, injeta falhas modificando o sistema operacional sobre o qual o sistema sob teste está rodando. A ferramenta foi desenvolvida para ser usada com o sistema operacional Linux , que é *open-source*, com isso os desenvolvedores podem reescrever algumas partes do Linux.

44

Assim eles conseguiram colocar uma parte do ComFIRM no sistema operacional, e a ferramenta é capaz de injetar falhas de dentro do sistema operacional. O objetivo da ComFIRM é injetar falhas de comunicação, pretende testar mecanismos de tolerância a falhas relacionadas a comunicação dentro de um sistema distribuído, uma vez que ela o faz através da troca ou da remoção das mensagens trocadas entre os nós do sistema. Quando um módulo, em algum nó do sistema chama uma função do sistema operacional para transmitir mensagens, essa função deve ser trocada por uma função do ComFIRM que irá alterar ou remover a mensagem a ser transmitida.

ComFIRM têm a vantagem de injetar falhas a partir do sistema operacional. Ele é capaz de fazê-lo com baixa perturbação do sistema sob teste, que roda com a mesma velocidade de execução. Têm como desvantagem a dependabilidade do sistema operacional Linux, ou seja o sistema alvo deve rodar sobre esse sistema operacional.

3.3.4. Doctor

Doctor *(Integrated Software Fault Injection Environment)* [Hsueh 1997], criado pela Universidade de Michigan, combina vários métodos para injetar falhas. Usa *time-outs*, *traps* e alteração de código para fazer a injeção de falhas. Para simular falhas de memória, Doctor usa *time-out*: ele programa um *timer*, e quando expira dispara uma interrupção, previamente programada no vetor de tratamento de interrupções. *Traps* são usados para simular falhas de processador. A vantagem é que time-outs e traps são recursos disponíveis em quase todos os sistemas, e assim a ferramenta pode ser reescrita para rodar em vários ambientes. Contudo, pelo fato de usar *time-outs* e *traps*, todos causam perturbação no fluxo de execução do sistema, a execução do sistema sob teste pode ficar bem diferente da que seria rodando em condições reais.

3.3.5. FIESTA

FIESTA *(Fault Injection for Embedded System Target Application)* [Krishnamurthy 1998], desenvolvida pela Universidade do Texas em Austin, é uma ferramenta de injeção de falhas projetada para o sistema operacional de tempo real *VxWorks*. A ferramenta depende

45

de um depurador para injetar falhas. Por exemplo, ele pode localizar uma instrução específica e colocar um *breakpoint* naquela instrução. FIESTA trabalha com falhas de linha de endereço quando está buscando uma instrução, falhas transientes de registradores e mutação aleatória de código executável. A vantagem é que a ferramenta baseia- se em software desenvolvido e testado, que é a versão modificada do depurador gdb, porém a desvantagem é que se o usuário decidir injetar uma falha na décima vez que ele atinge uma instrução, o sistema irá parar dez vezes no breakpoint.

3.3.6. FSOFIST-mp

A ferramenta FSOFIST-mp, está sendo implementada para validar a arquitetura Ferry-Injection-mp proposta nesta dissertação. A ferramenta não está totalmente desenvolvida, mas pode-se apresentar as seguintes características desta ferramenta. A ferramenta simula falhas de comunicação. Neste caso, o injetor geralmente é inserido entre a camada alvo e a camada inferior [Echtle 1992]. Os erros injetados têm por objetivo representar falhas possíveis de ocorrer em um sistema distribuído, e consistem geralmente na alteração, perda, duplicação ou retardo de mensagens transferidas pelo canal de comunicação. A vantagem desta ferramenta é que ela está sendo implementada para ser totalmente modular, facilitando a reusabilidade da ferramenta, é portátil e principalmente suporta testes multi-partes. A desvantagem é a intrusão dos testes na implementação sob teste quando pretende-se injetar falhas de memória ou processador.

3.3.7. Comparação entre as ferramentas

Cada uma das ferramentas apresentadas tem suas vantagens e desvantagens, assim torna-se importante uma comparação baseado em algumas características comuns, que estão apresentadas na Tabela 3.1.

46

TABELA 3.1: Comparação entre ferramentas de Injeção de Falhas.

Ferramenta	*Perturbação*	*Portabilidade*	*Tipos de Falhas*	*Sistema Alvo*
ORCHESTRA	quase nenhuma	fácil	falhas de transmissão de mensagem em um sistema distribuído	sistemas distribuídos
Xception	quase nenhuma	difícil	falhas de CPU, memória e barramento	sistemas baseados em processadores modernos
ComFIRM	baixa	média, depende da portabilidade do sistema operacional Linux	falhas de transmissão de mensagem em um sistema distribuído	sistemas distribuídos baseados no sistema operacional Linux
Doctor	média	fácil, porém o sistema deve conter traps e time-outs	falhas de processador e memória	qualquer sistema (mais comum sistemas didtribuídos)
FIESTA	baixa	depende do depurador	falhas de endereço, processador e memória	qualquer sistema
FSOFIST-mp	depende do tipode falha a ser injetada	fácil	falhas de transmissão de mensagem em um sistema distribuído	qualquer sistema (originalmente sistemas distribuídos

3.4. Arquitetura genérica para injeção de falhas

A partir de estudos das ferramentas de injeção de falhas desenvolvidas, foi realizada a formulação de uma arquitetura genérica para Injeção de Falhas [Hsueh 1997]. Esta arquitetura é apresentada na Figura 3.1.

Nesta arquitetura existe um injetor de falhas, que simula a presença de falhas no sistema sob teste. As falhas estão armazenadas numa biblioteca de falhas. Há também um monitor, que observa o comportamento do sistema alvo, bem como um gerador de carga de trabalho que irá transmitir comandos para o sistema sob teste executar, de acordo com a biblioteca de carga de trabalho. Os dados sobre o sistema são recolhidos pelo coletor de dados, e posteriormente podem ser analisados pelo analisador de dados. O controlador coordena a execução de todo o sistema [Hsueh].

47

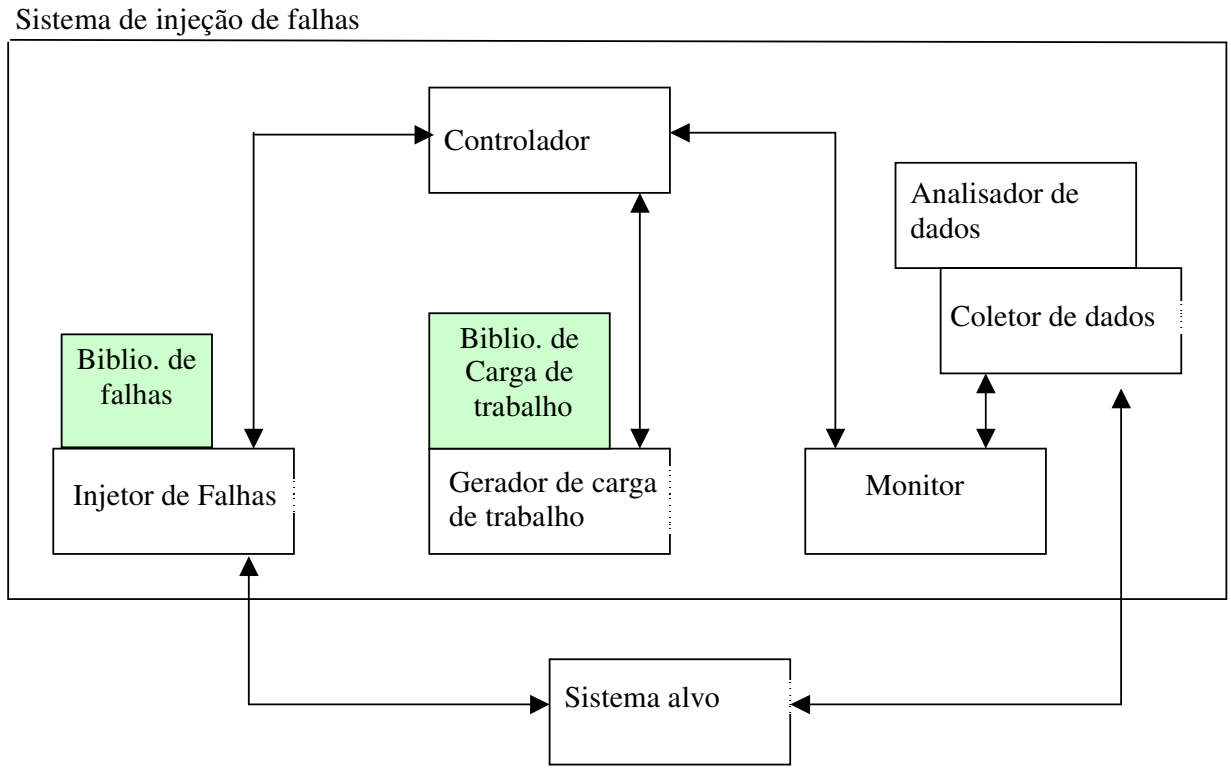


FIGURA 3.1. Diagrama de uma arquitetura genérica para Injeção de Falhas, adaptada a partir de [Hsueh]

3.5. Projeto ATIFS

O ATIFS (Ambiente de Testes por Injeção de Falhas por Software) é uma plataforma que possui um conjunto de ferramentas que apóiam as atividades de geração, execução e análise de resultados dos testes.

O ATIFS apóia a realização de dois tipos de testes: *testes de conformidade*, cujo objetivo é verificar se a implementação de um sistema está de acordo com a sua especificação e *testes por injeção de falhas*, cujo objetivo é verificar o comportamento da implementação em testes em presença de falhas introduzidas de maneira controlada durante a execução do sistema.

O processo de teste [Araújo] possui quatro partes:

48

- Geração de Testes (*Test Generation*): processo de produção de casos de teste a partir de uma especificação do protocolo;
- Seleção e Parametrização de Testes (*Test Selection and Parameterization*): onde se seleciona um subconjunto entre todos os casos de teste possíveis, escolhendo os valores para os parâmetros de cada caso de teste, para aplicação e validação de uma implementação;
- Execução de testes: nesta etapa os testes gerados e devidamente parametrizados são aplicados à implementação, sendo o resultado armazenado para análise posterior;
- Análise dos resultados (*Results Analysis*): processo no qual é verificado se o comportamento de uma dada implementação durante os testes foi o esperado de acordo com sua especificação.

A Figura 3.2 apresenta a arquitetura do ATIFS e em seguida é apresentado um resumo de cada subsistema que compõe o ATIFS. Neste trabalho o enfoque é dado ao subsistema SSE (Sistema de Suporte à Execução) na aplicação dos testes.

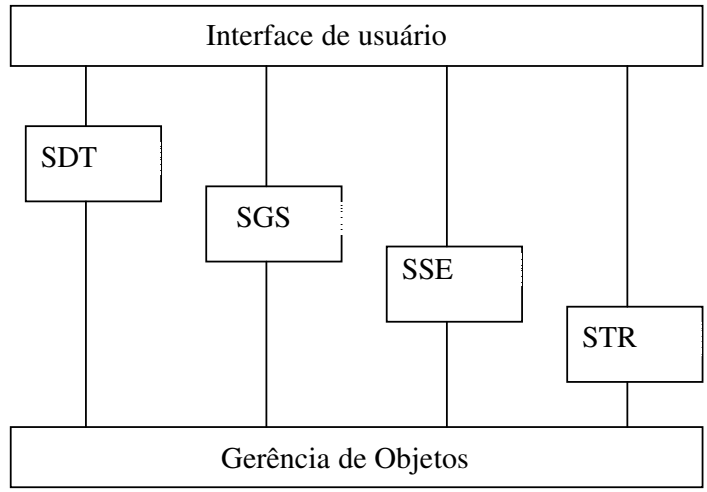


FIGURA 3.2. Arquitetura do ATIFS

3.5.1. Subsistema de Desenvolvimento de Testes (SDT)

O Subsistema de Desenvolvimento de Testes é o responsável pela geração de testes (abstratos) a partir do protocolo de comunicações especificado como máquina

49

finita de estado estendida (MFEE). Uma MFEE permite não só representar o aspecto controle, relativo à seqüência válida das interações, mas também o aspecto dados, referentes aos valores de parâmetros dessas interações, entre outras. O SDT é composto pelas seguintes ferramentas:

- Condado, responsável pela geração de testes a partir da MFEE, combinando diferentes métodos de testes que permitam cobrir tanto o aspecto controle quanto o aspecto dados [Araújo];
- GeraParm, responsável pela parametrização dos casos de testes, isto é, pela geração dos parâmetros das interações [Araújo];
- VerProp, que verifica se o modelo fornecido possui as propriedades requeridas pelos métodos de teste utilizados [Araújo].

3.5.2. Subsistema de Geração de Scripts (SGS)

O SGS permite a edição dos casos de teste criados pelo SDT e a geração de um script (conjunto de testes executáveis) contendo os casos de testes e eventualmente, as falhas, caso o usuário deseje realizar testes por injeção de falhas. Considera-se aqui a injeção de falhas de comunicação, afetando a troca de mensagens com a implementação em teste e que consiste em corromper, suprimir, duplicar ou atrasar as mensagens.

As falhas podem ser definidas manualmente pelo usuário, com o auxílio de caixas de diálogo e outros recursos da interface gráfica, ou automaticamente. Neste caso o SGS define o valor de seus atributos (mensagem a ser alterada, local da mensagem a ser alterado, se as falhas afetarão todas as mensagens ou só algumas, entre outras) aleatoriamente entre os possíveis valores. A definição automática de falhas e a possibilidade da reutilização de um conjunto de falhas aceleram o processo de preparação dos testes. A saída do SGS é um script em TCL que controlará a execução dos testes.

50

3.5.3. Subsistema de Suporte à Execução (SSE)

O Subsistema de Suporte a Execução lê o script gerado pelo Sistema de Geração de Script, inicia e controla a execução deste script. Durante o processo de execução é permitido a usuário monitorar os testes. O SSE contém a ferramenta FSOFIST (*Ferry-clip with SOftware Fault-Injection Support Tool*) [Araújo], que implementa a arquitetura Ferry-clip, proposta pela ISO para testes de protocolos de comunicação. Dentre as vantagens dessa arquitetura tem-se portabilidade (ela é facilmente configurável para os testes de diferentes protocolos executando em diferentes plataformas) e facilidade de extensão, como foi o caso do protótipo da FSOFIST estendido para englobar também os testes por injeção de falhas.

O SSE [Martins] interage com o sistema em testes (sistema alvo) de acordo com o script, enquanto coleta e armazena dados observados nos diversos nós do sistema alvo.

3.5.4. Subsistema de Tratamento dos Resultados (STR)

Este subsistema trata os dados coletados durante os testes e gera um relatório final contendo os resultados da validação. Este relatório será composto de gráficos, contendo os principais resultados da análise. As duas funções do STR são [Martins]:

- Análise do comportamento: verifica automaticamente se o resultado observado está de acordo com o que foi especificado.
- Obtenção de estatísticas: permite ao usuário definir os parâmetros de qualidade que se deseja obter (ex: fator de cobertura, latência), bem como o método de estimação a ser empregado para obtê-los. Serão obtidos os valores desejados e os erros associados à estimativa após a escolha e aplicação de um método.

51

CAPÍTULO 4

ARQUITETURA E FERRAMENTA PROPOSTA

4.1. Introdução

Este Capítulo descreve a arquitetura de testes proposta neste trabalho, ferry-Injection-mp, e os requisitos da ferramenta, FSoFIST-mp, para validação desta arquitetura, tendo como principais características à execução de testes de conformidade para sistemas multi-partes e a injeção de falhas por software.

A próxima Seção descreve os requisitos definidos para a ferramenta. A Seção 4.3 apresenta o sistema de desenvolvimento. Na Seção 4.4 são apresentados os aspectos de projeto, como: a arquitetura e a utilização de padrões de projeto. A Seção 4.5 apresenta o modelo de classes e a descrição das classes que o compõe, diagrama de seqüência e o diagrama de estados dos ferrys clips.

4.2. Requisitos

A ferramenta de suporte à execução dos testes é responsável pela ativação, interrupção, interação e monitoração do sistema em teste, exercendo o papel de estimulador e receptor dos eventos necessários à realização do teste. Um controle dos recursos como memória primária, secundária, serviços de comunicação e de temporização pode ser realizado. A preparação do sistema antes de cada teste e a restauração do mesmo para um estado estável após a aplicação do teste é sempre executada por esta ferramenta. Ela deve possuir características para se adaptar às condições da aplicação (IUT - Implementation Under Test) com um mínimo de esforço e de alterações, assim pode-se dizer que a ferramenta deve:

- Controlar a execução dos testes, isto é, permitir ao usuário inicializar, interromper, continuar ou finalizar o teste em execução;

52

- Monitorar o sistema em teste, ou seja, observar e armazenar as interações de entrada e saída que ocorrem durante os testes;
- Supervisionar os testes com relação aos recursos como memória disponível e acessível, aos serviços de comunicação, as temporizações;
- Preparar o sistema para os testes: carregar a IUT, o Passive Ferry - PF (com todos os componentes), o sistema de teste (Test Engine e o Active Ferry -AF), estabelecer conexão PF e AF, estabelecer conexão PF e IUT e restaurar o sistema (trazê-lo a um estado estável) após aplicação dos mesmos;
- Executar os testes;
- Simular a ocorrência de falhas dentro do sistema sob teste, de maneira a causar o mínimo de perturbação na execução do sistema alvo.

4.2.1. Componentes do Sistema de Teste

A arquitetura *ferry* prevê dois sistemas distintos, o sistema de teste e o sistema sob teste. O sistema de teste é responsável pelo controle dos testes, a seguir são apresentados os componentes que compõe este sistema:

Test Manager: este componente controla toda a operação do sistema de teste. Cabe a ele realizar as seguintes funções:

- efetivar a comunicação com o usuário, aguardando um sinal para iniciar aplicação dos testes, exibir a evolução dos testes, permitir a interrupção, a continuação e o encerramento dos testes pelo usuário;
- indicar o início e o término dos testes, controlando a execução das entidades do Sistema de Teste;
- gerenciar os *"ferry":* sinalizar a abertura de conexão entre AF e PF no início dos testes, checar a conexão de tempos em tempos, sinalizar o fechamento da conexão no fim dos testes;
- abortar a execução quando uma situação anormal é detectada.

Test Engine: deve acoplar as funcionalidades do Testador Inferior e Superior e ativar o sistema sob teste;

53

SIA: este módulo é responsável pela conversão dos dados da interface para um formato que a IUT entenda.

LMAP: deve trocar dados entre o AF e o PF.

FSM: deve controlar a comunicação entre o AF e o PF.

FIC: deve controlar a injeção de falhas.

4.2.2. Componentes do Sistema em Teste

O sistema sob teste é responsável pela execução dos testes propriamente dita, seus componentes estão descritos abaixo:

SIA: deve converter o formato da IUT para um formato compatível com o sistema de teste;

LMAP: deve trocar dados entre o AF e o PF;

FSM: deve controlar a comunicação entre o AF e o PF;

FIM: deve injetar as falhas propriamente dita.

IUT: implementação sob testes.

4.3. Sistema de Desenvolvimento

A ferramenta FSOFIST-mp (com Multi Partes) está sendo implementada em microcomputador do tipo PC, rodando Windows XP e utilizando a linguagem de programação Java [Sun]. Os seguintes fatores contribuíram para esta escolha:

- Java é uma linguagem orientada a objetos;
- Grande portabilidade;
- Disponibilidade de uma grande variedade de ambientes de desenvolvimento;
- Software livre.

A ferramenta está sendo implementada usando o jdk1.5.0 e o ambiente de desenvolvimento eclipse 3.0, que também é software livre.

O estudo das metodologias de desenvolvimento consistiu em identificar as principais metodologias existentes e escolher uma metodologia para ser utilizada neste

trabalho. As principais metodologias para desenvolvimento de software orientado a objetos encontradas foram [Douglas]: Booch, OMT- Técnica de Modelagem de Objetos (Object Modelling Technique), OOSE (Object Oriented Software Engineering).

A UML (Unified Modeling Language) foi escolhida como linguagem de modelagem por sua popularidade além de ser considerada como um padrão, e poder ser aplicada nas diferentes fases do desenvolvimento de um sistema, desde a fase de especificação dos requisitos até a fase de testes.

Neste trabalho foram utilizados dois tipos de diagramas para visualização e compreensão do sistema. O diagrama de classe foi utilizado para fazer a modelagem da visão estática do projeto do sistema e os diagramas de seqüência e estados foram empregados para a modelagem dos aspectos dinâmicos do sistema [Booch 2000].

4.4. Aspectos de Projeto

Com o objetivo de deixar a arquitetura Ferry Injection-mp totalmente modular, foi utilizado um sistema de padrões [Leme 2001]. Este sistema de padrões visa facilitar o desenvolvimento de novas ferramentas para injeção de falhas. A seguir é apresentada uma visão geral desses padrões:

O padrão Injetor de falhas é um padrão de arquitetura que visa solucionar o problema de como arquiteturar um programa para realizar a injeção de falhas. Quais seriam seus componentes, como se relacionariam e como seriam suas interfaces.

O padrão de arquitetura de injeção de falhas, foi estruturado em 7 subsistemas:

- *Ativador:* ativa a execução do sistema alvo para que ele possa ser testado em suas condições de funcionamento normais;
- *Injetor:* realiza a injeção propriamente dita das falhas dentro do sistema sob testes;
- *Monitor:* faz a monitoração do sistema alvo, para verificar se ele opera como o esperado;
- *Controlador:* controla os subsistemas acima, para que realizem suas atividades coordenadamente;

55

- *Interface com o usuário:* recebe as especificações do usuário para realização do experimento e devolve os resultados;
- *Gerenciador de falhas:* repositório de falhas que armazena as falhas a serem injetadas;
- *Gerenciador de dados monitorados:* armazena os resultados recebidos da monitoração do sistema sob testes.

Além desse padrão, existem dois padrões de projeto [Leme 2001] que objetivam resolver problemas localizados dentro da arquitetura do sistema. São os padrões injetor e monitor apresentado a seguir.

O injetor é um padrão de projeto que sugere uma estrutura de objetos que se comunicam com o sistema sob teste e simulam uma determinada falha dentro dele. Esta estrutura tem três componentes:

- Gerenciador de Injeção: esse componente controla o processo de injeção em si. Ele instancia os injetores, descritos a seguir, de acordo com o tipo de falha injetada.
- Injetor: é instanciado pelo gerenciador de injeção para cuidar da injeção de falha em específico. É definida uma classe abstrata, injetor, e a partir dessa classe, são criadas subclasses concretas que implementam tipos de falhas diferentes. Porém ele não se comunica diretamente com o sistema sob teste, isto é realizado através do injetor físico.
- Injetor físico: é o componente que se comunica diretamente com o sistema sob teste através de primitivas para comunicação.

O monitor é também um padrão de projeto que procura fornecer uma solução para o problema de como criar uma estrutura que faça a monitoração do sistema sob teste. A estrutura apresenta três componentes:

- Gerenciador de Monitoração: este componente coordenaria o processo de monitoração.Para cada aspecto do sistema alvo, ele instancia um sensor para cada tipo de dado a ser obtido.
- Sensor: objeto encarregado de monitorar um determinado aspecto do sistema sob teste. É definida uma classe abstrata, sensor, que define uma interface comum para cada tipo de sensor. A partir dessa classe, são criadas as subclasses concretas que especificam sensores para cada tipo de aspecto monitorado.

56

- Sensor físico: objeto que fazem a comunicação dos sensores com o sistema sob teste.

Com isso, para que a arquitetura ficasse totalmente padronizada, foi utilizado o padrão de interface *Bridge*. Esse padrão tem como função desacoplar uma abstração de interface de sua implementação para que ambos possam variar independentemente [Gamma 1994].

Baseado nestes padrões de software foi realizado, neste trabalho, o mapeamento desses padrões à arquitetura Ferry Injection-mp. A Tabela 4.1 apresenta esse mapeamento.

TABELA 4.1. Mapeamento dos componentes dos padrões para os componentes da Ferry Injection-mp.

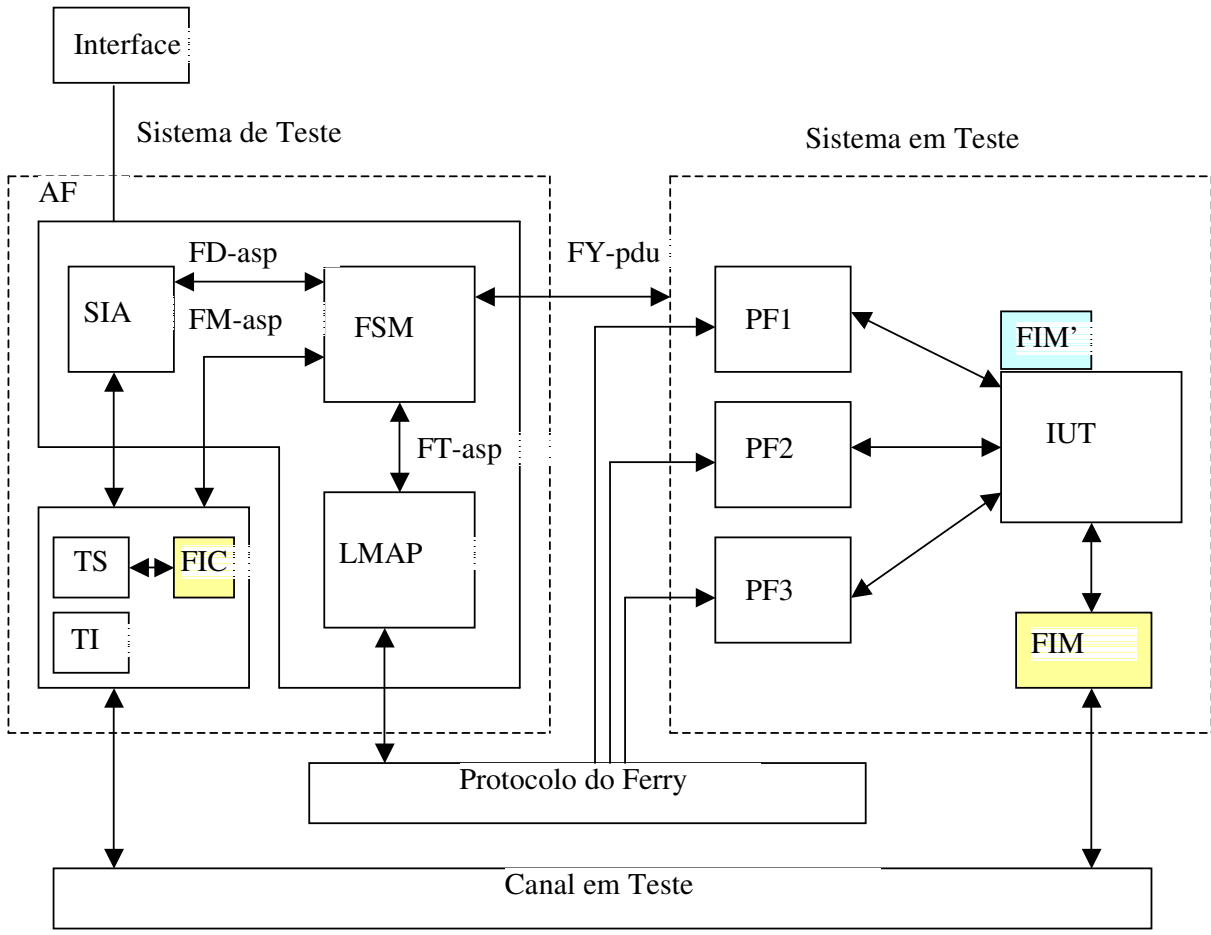
Padrão	**Componente no padrão**	**Ferry Injection-mp**
Injetor de falhas	Controlador	Test Manager
Injetor de falhas	Ger. de Falhas	Arquivo XML
Injetor de falhas	Ger. de dados monitorados	Log de saída
Injetor	Injetor e Ger. de injeção	FIC
Injetor	Injetor Físico	FIM
Monitor	Gerenciador de Mon., sensor.	
Monitor	sensor físico	Parte do PF
Ativador	Ger. de ativação e ativador	Test Engine
Ativador	Ativador Físico	Parte do PF (que não é injeção de falhas)
Bridge	Interface	Interface

A partir da utilização desses padrões e baseado na arquitetura Ferry Injection, a arquitetura Ferry Injection-mp foi proposta com o diferencial para suportar testes multi-partes que é um requisito fundamental para testes em sistemas espaciais. A Figura 4.1 apresenta a arquitetura Ferry Injection-mp. Nesta figura foram representados 3 PFs, em função da aplicação utilizada para análise da arquitetura, porém a arquitetura suporta n PFs. Outra particularidade apresentada nesta arquitetura é o módulo FIM' que fica junto da IUT, isto deve-se ao fato de os tipos de falhas de processador e falhas de memória são comuns em sistemas espaciais e visando poder injetar falhas neste dois casos foi incluído o módulo FIM'.

57

As vantagens de se usar essa arquitetura são:

- Modularidade: permite que as tarefas realizadas por cada componente sejam alteradas individualmente;
- Centraliza a comunicação entre os componentes em um único componente, o Test manager;
- Perturbação menor possível, porque apenas quem precisa se comunicar com o sistema sob teste, é que realiza essa comunicação;
- Possibilidade de executar testes e injeção de falhas em paralelo.



SAI: Service Interface Adapter
FSM: Finite State Machine
LMAP: Lower Mapping Module

FIGURA 4.1. Arquitetura Ferry Injection-mp

58

4.5. Modelo de Classes

Esta Seção apresenta o modelo de classes e a descrição das classes que compõem a ferramenta FSOFIST-mp que está sendo implementada para validar a arquitetura Ferry Injection-mp que foi proposta neste trabalho.

A Figura 4.2 apresenta a visão estática da FSOFIST-mp através do diagrama de classes. Este diagrama mostra as classes da ferramenta que está sendo implementada e o relacionamento entre elas. Para facilitar a visualização no diagrama as estruturas de dados não são listadas. As classes apresentadas na Figura 14 descrevem apenas os métodos.

59

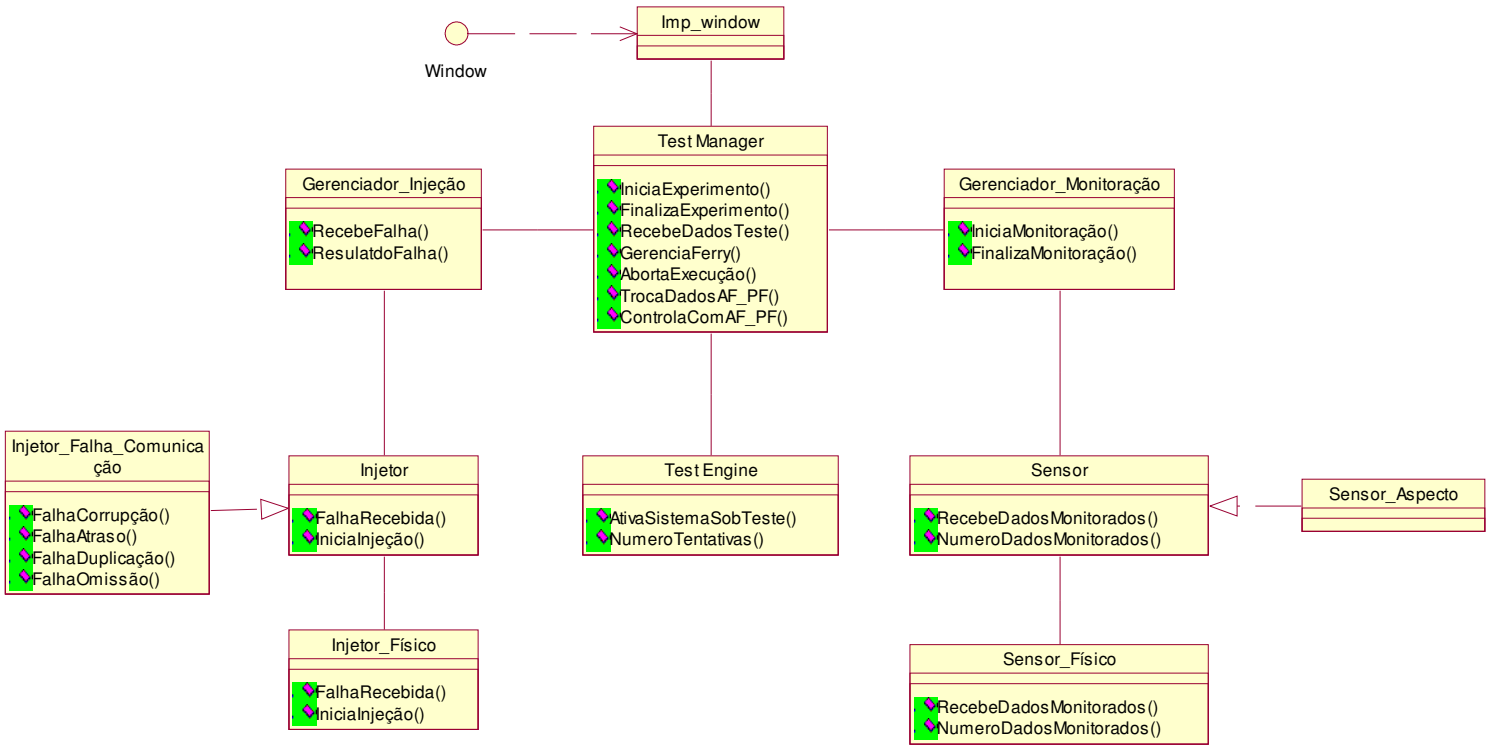


FIGURA 4.2. Diagrama de Classes da FSOFIST-mp

60

4.5.1. Descrição das Classes

Esta subseção descreve as classes e os métodos que estão sendo utilizados na FSOFIST-mp.

4.5.1.1. Classe Test_Manager

Esta classe gerencia todos os componentes do sistema além de invocar a interface do usuário. Os métodos desta classe estão descritos abaixo:
IniciaExperimento(): Este método é utilizado para iniciar os testes e o experimento de injeção de falhas sobre um sistema em teste.

FinalizaExperimento(): Este método é usado para finalizar os testes.

RecebeDadosTeste(): Este método é invocado pela interface para passar os dados fornecidos pelo usuário.

GerenciaFerry(): Este método é utilizado para abertura de conexão entre AF e PF e checagem de tempos em tempos da conexão.

AbortaExecução(): Método utilizado para abortar a execução dos testes caso tenha sido lançada alguma exceção.

TrocaDadosAF_PF(): Este método é responsável pela troca de dados entre o Active Ferry e o Passive Ferry.

ControlaComAF_PF(): Este método controla a comunicação entre o Active Ferry e o Passive Ferry.

61

4.5.1.2. Classe Test_Engine

Esta classe é responsável pela ativação do sistema sob teste. E seus métodos são:

AtivaSistemaSobTeste(): Este método é responsável por ativar o sistema em teste. Retorna TRUE am caso de sucesso e FALSE caso não tenha conseguido ativar o sistema.

NumeroTentativas(): Este método retorna o número de tentativas de ativação do sistema alvo.

4.5.1.3. Classe Gerenciador_Monitoração

Esta classe coordena o processo de monitoração do sistema. Para cada aspecto alvo ele instancia um sensor. Neste caso é analisado apenas um aspecto.

IniciaMonitoração(): Este método sinaliza que a monitoração do sistema em teste deve começar.

FinalizaMonitoração(): Método que indica que a monitoração do sistema foi finalizada.

4.5.1.4. Classe Sensor

Classe abstrata que define uma interface comum para cada tipo de sensor. A partir dessa classe são criadas classes concretas que especificam os sensores para cada tipo de aspecto.

62

4.5.1.5. Classe Sensor_Físico

Esta classe faz a comunicação dos sensores com o sistema sob teste.

RecebeDadosMonitorados(): Método invocado pelo gerenciador de monitoração para passar ao Test Manager os dados obtidos a partir da monitoração do sistema sob teste.

NumeroDadosMonitorados(): Retorna a quantidade de dados monitorados que foram armazenados.

4.5.1.6. Classe Gerenciador_Injeção

Esta classe controla o processo de injeção propriamente dito. Instancia os injetores de acordo com o tipo de falha a ser injetada.

RecebeFalha(): Recebe a especificação de uma falha a ser injetada no sistema sob teste.

ResultadoInjeção(): Este método retorna o resultado da injeção conforme especificação.

4.5.1.7. Classe Injetor

Esta classe é instanciada pelo gerenciador_injeção para cuidar da injeção de uma falha específica. Esta classe é abstrata e cada subclasse dela é implementada para trabalhar com um tipo de falha.

4.5.1.8. Classe Injetor_Falha_Comunicação

Classe criada para suportar injeção de falhas de comunicação.

63

Falha_Omissão(): Este método identifica o pacote a ser suprimido e retira-o da comunicação com o sistema sob teste.

Falha_Atraso(): Este método aloca memória em um buffer onde mantém uma cópia do pacote, omitindo-o temporariamente e deixando que o próximo pacote seja processado pela IUT. Após um determinado número de pacotes terem sido processados, este pacote é liberado.

Falha_Duplicação(): Este método identifica o pacote a ser duplicado e o tempo de atraso em número de pacotes trocados até que o pacote seja liberado novamente.

Falha_Corrupção(): Esta corrupção deve ser feita bit a bit. Além da identificação da mensagem é necessária uma máscara de bits identificando quais bits devem ser modificados no buffer antes do seu envio.

4.5.1.9. Classe Injetor_Físico

Esta classe é o componente que se comunica diretamente com o sistema sob teste e injeta a falha propriamente dita.

FalhaRecebida(): Verifica se o gerenciador de injeção já recebeu a especificação da falhas a ser injetada.

IniciaInjeção(): Indica que a injeção deve ser realizada de acordo com a especificação recebida.

4.6. Diagrama de estados dos ferrys clips.

Esta subseção apresenta o diagrama de estados dos principais componentes da arquitetura, o Active Ferry e o Passive Ferry.

64

4.6.1. Diagrama de estados da FSM do AF

A máquina de estados do AF pode estar em três estados possíveis: ocioso, conectando e conectado. A figura 4.4 mostra estes estados e suas possíveis transações.

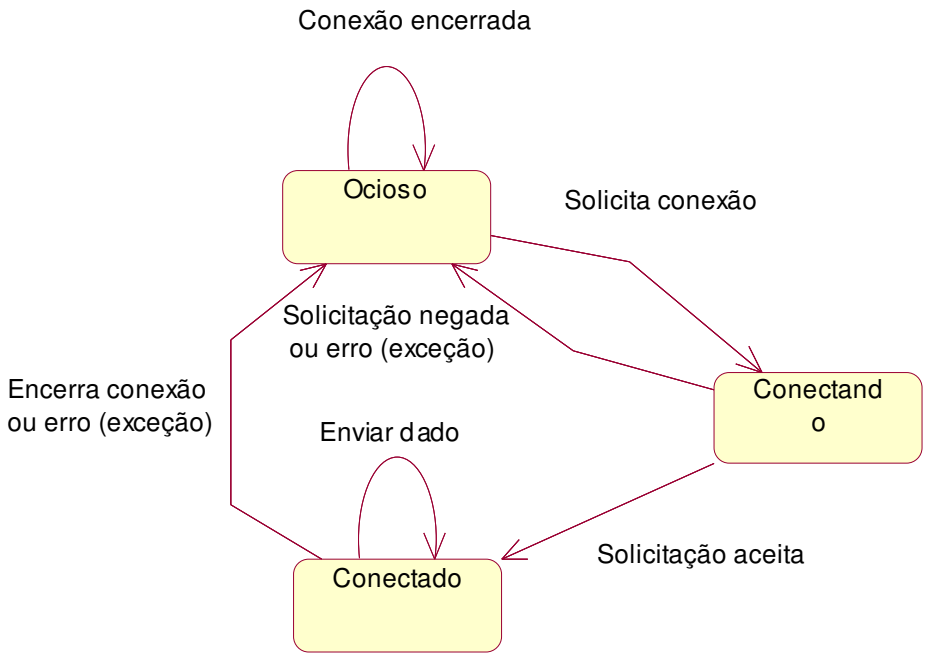


FIGURA 4.4. Diagrama de estados do AF

4.6.2. Diagrama de estados da FSM do PF

Este diagrama apresenta uma das visões do comportamento dinâmico do PF. A Figura 4.5 mostra este comportamento.

65

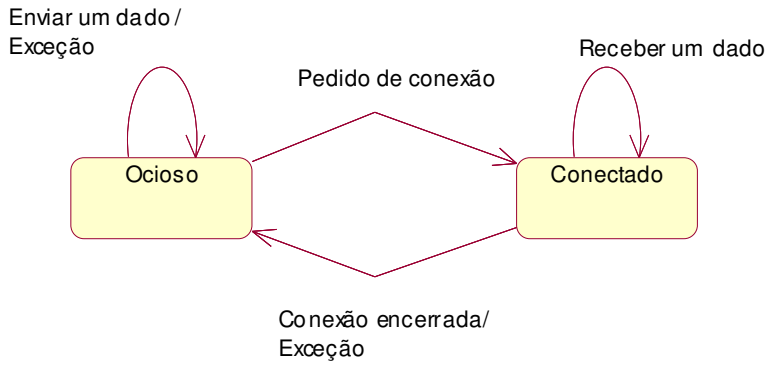


FIGURA 4.5. Diagrama de estados do PF

66

CAPÍTULO 5

ANÁLISE ARQUITETURAL

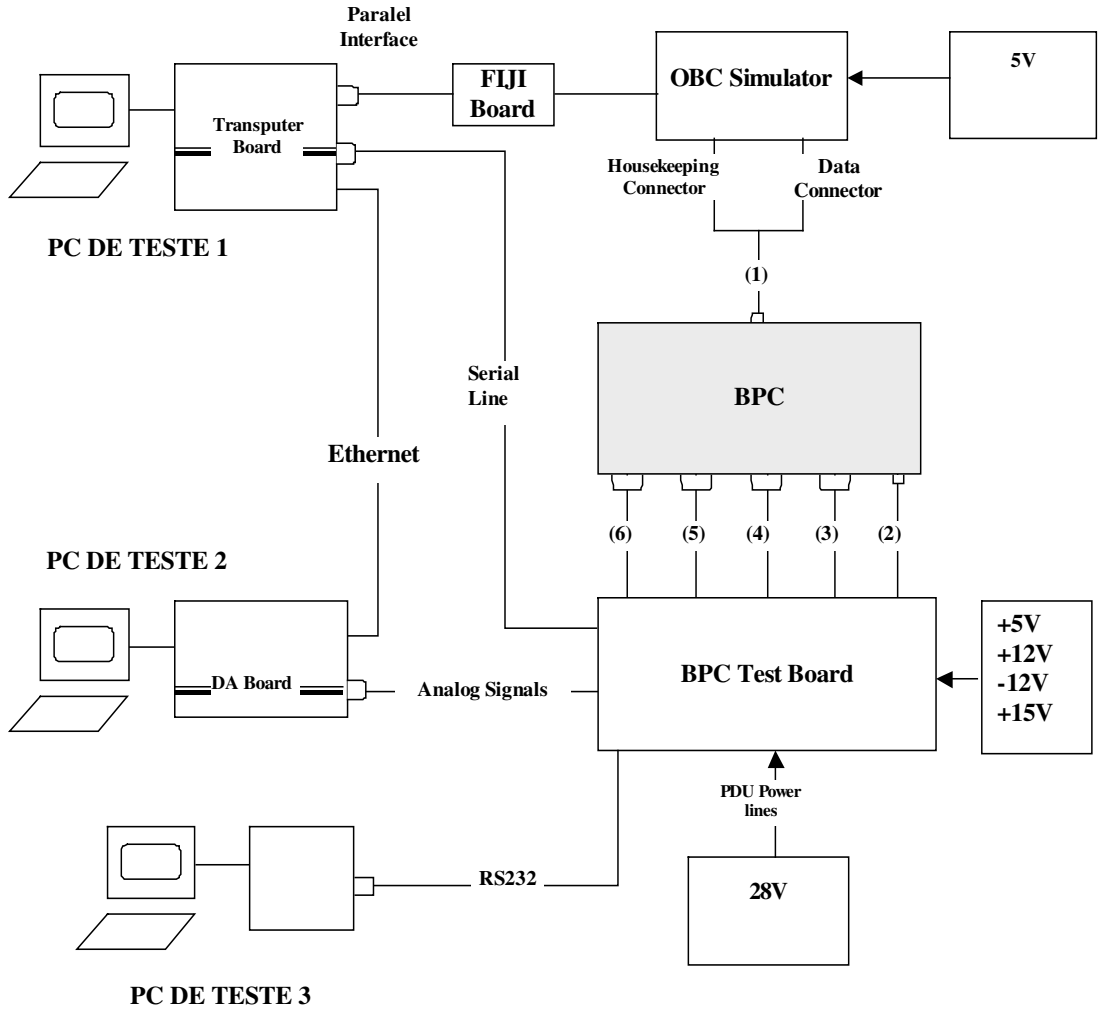
5.1. Introdução

Este Capítulo apresenta uma análise da arquitetura Ferry-Injection-mp para sistemas espaciais com o objetivo de validá-la. A próxima Seção descreve a arquitetura do Equipamento de Testes (ETBPC) do Computador dos Experimentos Brasileiros (BPC) do Microssatélite Franco-Brasileiro (FBM), que será utilizado na análise. A Seção 5.3 aborda a análise das propriedades da arquitetura proposta.

5.2. Arquitetura do Equipamento de Testes do Estudo de Caso.

O equipamento de testes do BPC é responsável por prover os recursos necessários para desenvolvimento e testes do hardware e software do BPC [FBM]. A Figura 5.1 mostra esta arquitetura e a Figura 5.2 mostra o Software do Equipamento de Testes. O software do Equipamento de Testes está distribuído no PC de teste 1, que contém a placa de desenvolvimento do *Transputer*, no PC de Teste 2, que contém a placa AD/DA, no PC de Teste 3 e no Simulador do Equipamento denominado *OBC (On Board Computer).*

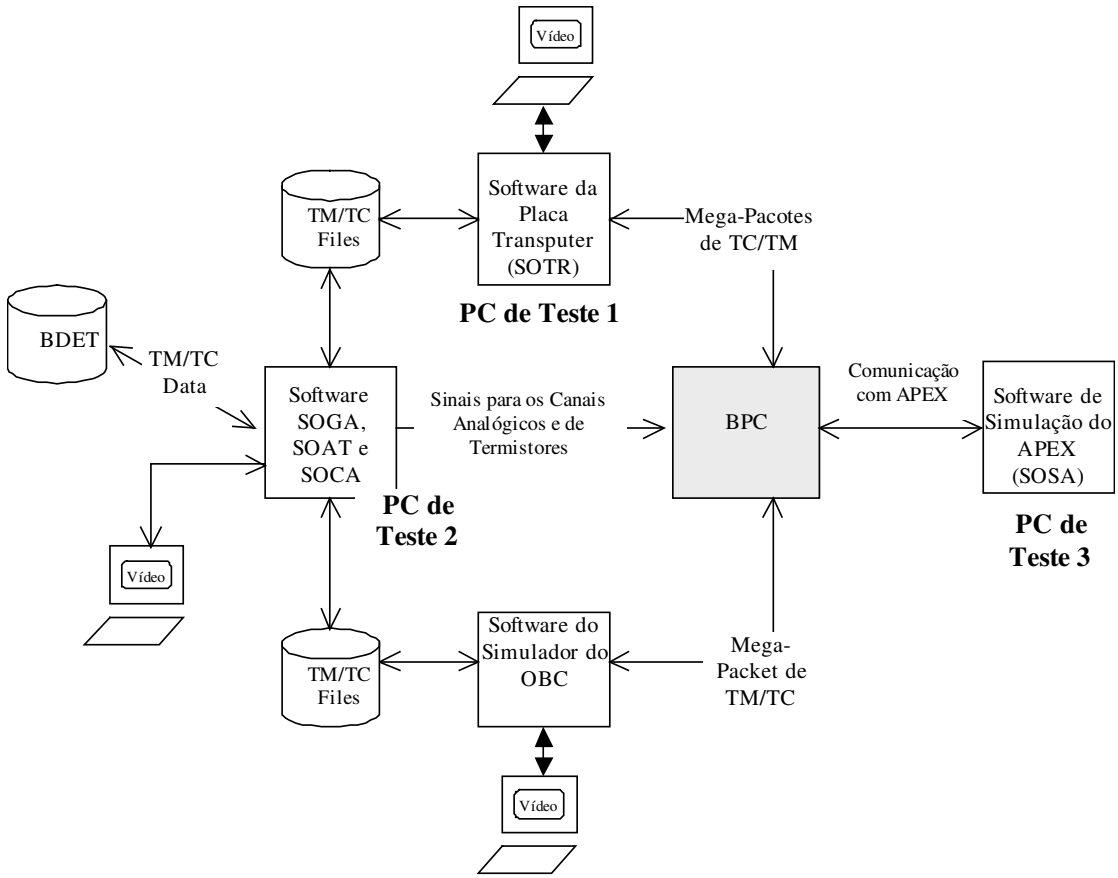
67



Onde: (1) - Conector de comunicação do OBC (Data e Housekeeping connector)
(2) - Conector de sinais de testes
(3) - Conector de sinais com os experimentos PDP e CPL
(4) - Conector de sinais com os experimentos CBEMG, APEX e FLUXRAD
(5) - Conector com a PDU
(6) - Conector de distribuição de potência para os experimentos PDP, CPL, CBEMG e FLUXRAD

FIGURA 5.1 – Arquitetura do Equipamento de Testes do BPC.

68



Onde : BDET : Banco de Dados do Equipamento de Testes
SOGA : Software para Geração de Pacotes de TM dos Experimentos
SOAT : Software de Análise de Telemetria
SOCA : Software de Controle da Placa AD/DA
SOSA : Software de Simulação do APEX
SOSI : Softwade do Simulador do OBC
SOTR : Software da Placa do Transputer
TM/TC Files : Arquivos com pacotes de TM e TC

FIGURA 5.2. Software do Equipamento de Testes

A comunicação com o BPC é feita através da linha serial denominada OsLink da placa de desenvolvimento para transputer ou do OBC.

69

5.2.1. Software do PC de Teste 1 (Simulador do OBC)

Este software tem como objetivo simular a operação do OBC, possibilitando a execução dos testes funcionais do BPC, antes da integração com o OBC real. Ele é implementado utilizando um ambiente de desenvolvimento integrado para a linguagem de programação Occam, que permite editar, compilar, ligar, configurar e carregar os programas na placa de desenvolvimento e no BPC.

As principais funcionalidades do software de simulação do OBC são:

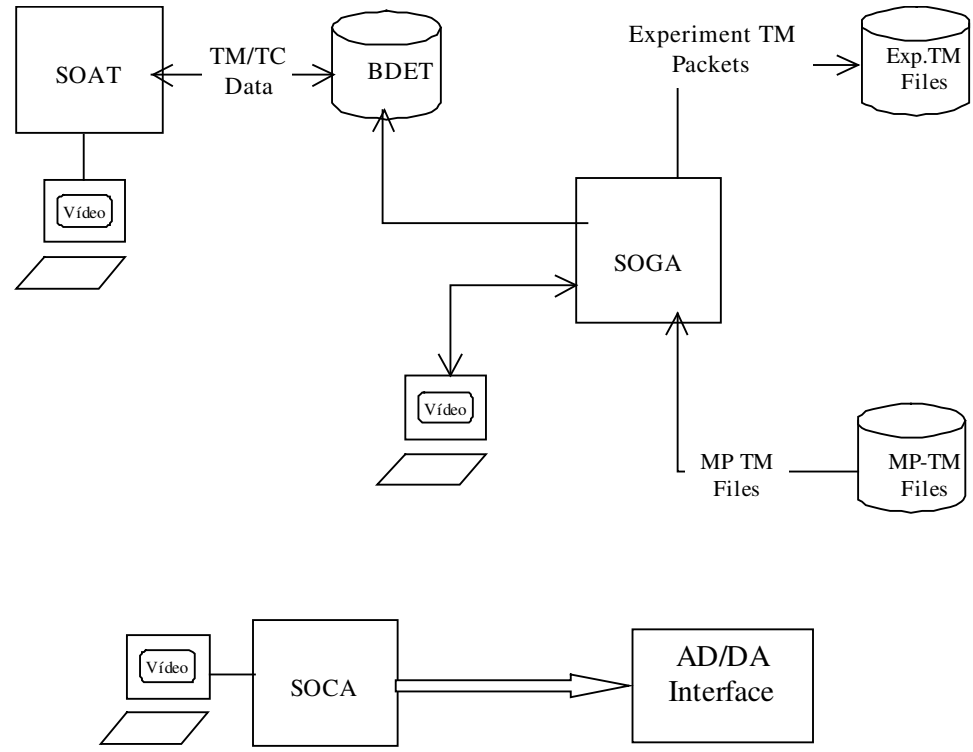
- Recepção de Mega-Pacotes de TM do BPC;
- Armazenamento de Mega-Pacotes de TM recebidos do BPC em disco;
- Visualização de mensagens de operação do software do BPC e Simulador do OBC;
- Armazenamento de mensagens de operação do software do BPC em disco;
- Envio de TC imediatos ou temporizados para o BPC.

O software de Simulação do OBC se comunica com o BPC através de uma linha serial OsLink.

5.2.3. Software do PC de Teste 2

A Figura 5.3 descreve a arquitetura dos softwares executados no PC de Teste 2 (SOGA, SOAT e SOCA) e as próximas subseções descrevem suas funcionalidades. Eles são implementados usando o ambiente de programação Visual C++ 6.0.

70



Onde: SOAT : Software de Análise de Telemetria
SOGA : Software para Geração de Pacotesde TM dos Experimentos
SOCA : Software de Controle da Placa AD/DA

FIGURA 5.3. Software do Equipamento de Teste do BPC

5.2.3.1. Programa SOGA

Este software lê os arquivos que contêm Mega-Packets de TM recebidos do BPC, verifica a consistência dos dados, extrai os pacotes de telemetria dos experimentos APEX, PDP, CPL, CBEMG e FLUXRAD e armazena em arquivos separados.

Para cada arquivo de Mega-Packets analisado, é gerado um conjunto de arquivos com os Pacotes de TM dos experimentos encontrados.

71

5.2.3.2. Programa SOAT

Este software tem como objetivo verificar a consistência dos dados transmitidos pelo BPC, armazenar em um banco de dados e fornecer recursos para sua visualização. Estas tarefas não são executadas em tempo real.

Os Arquivos contendo Mega-Packets de TM são gerados pelo software da placa de desenvolvimento do transputer ou pelo Simulador do OBC, dependendo da configuração de teste.

As principais funcionalidades do software de análise de TM são:

- Verificar a consistência dos pacotes de TM;
- Armazenamento de pacotes de TM no banco de dados do Equipamento de Testes;
- Visualização de dados de telemetria de acordo com o tipo de pacote de TM.

5.2.3.3. Programa SOCA

Este software tem com objetivo simular as entradas analógicas do BPC e permitir executar as funções descritas a seguir :

- Define o valor das saídas dos canais analógicos da placa AD/DA;
- Teste do controle do experimento CPL no modo Surviving;
- Teste do controle do experimento CPL no modo Normal;
- Simulação do FFT do experimento PDP.

5.2.4. Software do PC de Teste 3

Este software tem como objetivo simular o protocolo de comunicação BPC/APEX através da linha de comunicação serial RS422. A comunicação é feita através da interface COM1 do PC.

72

Os comandos recebidos do BPC e os dados enviados para o BPC são mostrados no vídeo.

Existem duas versões do software simulador do APEX: SimAPEX1 e SimAPEX2. A versão SimApex1 sempre transmite dados em resposta ao comando *Transmite Data* e a versão SimApex2 alterna a transmissão de dados com a resposta *No Data*.

A próxima Seção apresenta a aplicação da arquitetura Ferry-Injection-mp baseado neste estudo de caso.

5.2. Aplicação da Ferry-Injection-mp no estudo de caso

Nesta Seção serão apresentados alguns aspectos importantes de como a arquitetura Ferry_injection –mp contribuirá para os testes de sistemas espaciais cobrindo a maior parte das funcionalidades existentes neste tipo de sistemas.

5.2.1. Arquitetura com suporte a vários canais de comunicação

Em aplicações espaciais é comum termos mais de um canal de comunicação, comunicando-se com a IUT, devido a alta taxa de dados científicos que são transmitidos. Uma das características inerentes que deve ter na arquitetura é a possibilidade de ter vários canais de comunicação,e no exemplo do BPC tem-se três canais de comunicação comunicando-se simultaneamente com a IUT.

Para suprir esta característica foram projetados na arquitetura vários canais de comunicação entre o AF e o PF. Estes canais podiam ter as seguintes configurações: vários AF's e vários PF's ou um AF para vários PF's, nesta arquitetura foi escolhido ter apenas um AF por causa da dificuldade na sincronização de vários Active Ferry's.

Caso haja a necessidade de aumentar o número de canais físicos de comunicação entre o sistema de teste e o sistema sob teste basta acrescentar um PF.

73

5.2.2. O injetor de falhas

A proposta inicial da arquitetura era suprir os testes de funcionalidades e injeção de falhas de comunicação, porém ao estudar os sistemas espaciais verificou-se a necessidade de injetar outros tipos de falhas como falhas de memória e de processador.

Assim foi previsto que uma parte do FIM (FIM') ficaria junto com a IUT, para poder injetar falhas de memória e de processador. A desvantagem desta solução é a intrusividade que os testes têm na implementação sob testes.

Um estudo feito em [Rela] que trata da injeção de falhas durante o processo de boot, mostra o uso da técnica boundary-scan para injetar falhas. As falhas são injetadas em processador real, usado em missões espaciais. O artigo sugere como detectar erros propagados para aplicações em decorrência a falhas durante o processo de boot. Durante os experimentos somente duas falhas levaram erros a se propagar, causando *crash* da aplicação.

Quanto ao canal de comunicação para injetar as falhas, no ATIFS atual, as falhas só podem ser aplicadas no meio físico, em um determinado canal de comunicação físico.

Ao fazer um exercício de modelagem, verificou-se que as falhas poderiam ser injetadas em um canal lógico (não só nas linhas físicas). Um canal lógico poderia ser representado por variáveis de programa. Dessa forma, poderia-se injetar falhas de memória e de processador (se forem aplicadas sobre os valores de variáveis previamente definidas, pertencentes a canais lógicos). Mesmo as falhas de comunicação, não se limitariam a um canal físico.

Porém essa solução poderá apenas ser utilizado caso tenha-se acesso a variáveis do programa. O acesso não precisa ser irrestrito, mas a um conjunto mínimo para permitir que a implementação possa ser possível de ser testada.

5.2.3. Visualização de mensagens em disco

Está funcionalidade está sendo suprida pelo padrão monitor que é responsável por armazenar todas mensagens trocadas no sistema.

74

5.2.4. Dados dos testes e banco de dados

A intenção é deixar a arquitetura totalmente modular com o objetivo de alterar o mínimo possível de cada módulo, caso haja necessidade. De acordo com o estudo de caso apresentado um dos tipos de dados de testes são arquivos de telemetria e telecomando que são trocados entre o BPC e os subsistemas de testes. Na arquitetura atual o sistema tem banco de dados para armazenar esses arquivos.

Para ferramenta que está sendo desenvolvida optou-se por gerar um arquivo XML (eXtensible Markup Language) baseado no Test Profiler da OMG, que também apresenta uma estrutura modular**,** para gerar os casos de testes com dados passados pelo usuário através da interface e a saída também é um arquivo XML podendo este ser facilmente uma entrada para um banco de dados, um aplicativo excel, etc. O arquivo XML está demonstrado no Apêndice A

5.3. Vantagens da Arquitetura

A arquitetura apresenta várias vantagens para aplicações em sistemas espaciais como:

- Reusabilidade – com o avanço tecnológico na área espacial, as aplicações estão evoluindo rapidamente e com isso torna-se inviável uma arquitetura de testes para cada aplicação. Com a utilização da arquitetura uma vez que mude a IUT, apenas o PF deve ser alterado.
- Modularidade – a arquitetura é totalmente modular, com a utilização de padrões arquiteturais e de projeto, uma vez que um dos módulos precise ser alterado, apenas esse módulo específico precisa de tratamento.
- Portabilidade – fácil portabilidade.
- Cobertura de testes – cobrem os testes de conformidade e testes de injeção de falhas de comunicação, memória e processador.

75

CAPÍTULO 6

CONCLUSÕES E TRABALHOS FUTUROS

Conforme foi dito na Introdução desta dissertação, o objetivo principal deste trabalho era disponibilizar ao INPE uma arquitetura de testes para sistemas espaciais de modo a permitir reusabilidade, portabilidade e suporte a vários canais de comunicação. Pode-se dizer que o objetivo foi alcançado. A arquitetura ferry-Injection-mp foi criada e a ferramenta FSoFIST-mp está em implementação. Munidos destes dois recursos, os sistemas espaciais desenvolvidos poderão ser avaliados com maior precisão no funcionamento na presença de falhas.

Como a ferramenta FSoFIST-mp ainda está em desenvolvimento, foi realizado uma análise da Ferry-Injection-mp baseado em um sistema real do INPE. Nesta análise pôde observar que o objetivo de adaptar uma arquitetura para sistemas espaciais que cobrisse testes multi-partes, que fosse modular, portátil e fácil extensão, foi alcançado.

Atualmente para cada sistema desenvolvido no INPE é necessário uma arquitetura nova e conseqüentemente uma ferramenta de testes, com a arquitetura Ferry-Injection e posteriormente a ferramenta FSOFIST-mp, esse problema não ocorrerá.

A maior dificuldade encontrada neste trabalho foi com relação a localização do injetor de falhas, porque não foi possível encontrar um método que pudesse cobrir as injeções de falhas de memória e de processador sem intrusividade no sistema alvo. Assim como trabalho futuro fica a aplicação da técnica boundary-scan para injeção de falhas no processador.

Outro trabalho futuro é o término do desenvolvimento da ferramenta e aplicabilidade em um sistema real para sistemas espaciais. Além de estender a arquitetura para testes de interoperabilidade.

76

REFERÊNCIAS BIBLIOGRÁFICAS

Araujo, M. FSOFIST – Uma ferramenta para teste de protocolos tolerantes a falhas, IC-UNICAMP, Campinas, 2000

Barcelos, Patrícia P. A.; Leite, Fábio O.; Weber, Taisy Silva. "Implementação de um Injetor de Falhas de Comunicação". Anais do SCTF'99 – VIII Simpósio de Computação Tolerante a Falhas. Campinas, Brasil, Julho/99.

Booch, G.; UML, guia do usuário. Campus, 2000.

Carreira, J.; Madeira, H.; Silva, J.G. "Xception: Software Fault Injection and Monitoring in Processor Functional Units". 5th IFIP International Working Conference on Dependable Computing for Critical Applications. Urbana-Champaing, EUA, 1995.

Chanson, S. T; Voung, S.; Dany, H. "Multi-party and interoperability testing using the Ferry Clip approach", Computer communications vol 15, no 3, April 1992.

Dawson, S.; Jahanian, F.; Mitton, T. "ORCHESTRA: A Fault Injetion Environment for Distributed Systems".

Douglas, B. P. Real-Time UML- Developing Efficient Objects for Embedded Systems. Addison Wesley, 1998.

Echtle, K.; M. Leu. The EFA Fault Injection for Fault-Tolerant Distributed testing. Proc. Worshop on Fault-Tolerant Paralled and Distributed Systems, Amherst, USA, 1992.

FBM-XX-BC-07-5001-INPE, " French-Brasilian Microsatellite FBM". Dezembro, 2001.

77

Fischer, K. P.; GmbH, T. “Position Statement to IWPTS 1989” - Participant’s Proceedings 2 nd International Workshop on Protocol Test Systems, Berlim (West), Germany. October 3-6, 1989.

Gamma, E., Johnson, Johnson R.; Helm, R., Vlissides, J.; *“Design Patterns: Elements of Reusable Object-Oriented Software”*. Addison Wesley, 1994.

Gunnelo, U.; Karlsson, J.; Torire, J. “Evaluation of Error Detection Schemes using Fault Injection by Heavy-Ion Radiation”. Proc. FTCS-19, junho/1989, páginas 60-74.

Hsueh, Mei-Chen; Tsai, Timothy; Iyer, Ravishankar. “fault Injection Techniques and Tolls”. IEEE Computer, Abril/1997, páginas 75-82.

ISO TC97;SC21, IS 9646. “OSI Conformance Testing Methodology and FrameWork”, ISO 1991.

Krishnamurthy, N.; Jhaveri, V.; Abraham, J. “A Design Methodology for Software Fault Injection in Embedded Systems”. Proc of the 1998 IFIP International Workshop on Dependable Computing and its Applications. Johannesburg, Africa do Sul, Jan 1998, páginas 237-248.

Leme, Nelson G. M.; “Um sistema de padrões para injeção de falhas por software” - IC/UNICAMP – Agosto, 2001.

Madeira, H.; Furtado, P.; Silva, J.G. “RIFLE: A general purpose Fault Injector System”. Research Report, DEE-UC-007-91, Novembro/1991.

Martins, E. “ATIFS: um Ambiente de Testes baseado em Injeção de Falhas por Software” – Relatório Técnico DCC-95-24 – UNICAMP, dez 1995.

Pressman R.S. “Software Engineering: a Partitioner’s Approuch.”, McGraw-Hill, 4ª edição, 1997.

78

Rela, Mário Z.; Cunha, J. C.; Silva, L. F., “On the Effects of Errors During Boot” LADC, 2005.

Sun – Disponível na World Wide Web: http://java.sun.com/

Tretmans, J.; Belinfante, A.. “Automatic testing with formal methods”. Proc. EuroStar’99:7th. European Conference on Software testing, In proceedings of the Conference on Software Testing, Analysis and Review. EuroStar’99, November, 1999.

79

APÊNDICE A

Arquivo XML

```
<!--TesteSuite: conjunto de casos de teste -->
<TesteSuite tool="tool_name">
<!--TestCase: Pegando dados necessários para a execução de teste sem injeção de falhas
-->
                <TestCase name="testcase_name" objective="test case target">
                                        <Input>
                                    <Event name="init">
                                                <Param name="get_host_by_name">hostname</Param>
                                                <Param name="get_protocolo_by_name">tcp</Param>
                                                <Param name="get_protocolo_by_name">rs422</Param
                                                <Param name="get_service_by_name">dev</Param>
                                                <Param name="get_port_AF">1971</Param>
                                                <Param name="get_port_Experimento">1973</Param>
                                            </Event>
                                <Event name="Pacote_1">
                                                        <!--Mensagem de Controle -->
                                                    <Param name="TestCase">1</Param>
                                                <Param name="FluxoControl">1</Param>
                                                        <Param name="Modo">1</Param>
                                                <Param name="Id_Conexao">1</Param>
                                                <Param name="Campo_extra"> </Param>
                                                                <!--Mensagem de Dados -->
                                                            <Param name="Id">1</Param>
                                                        <Param name="Id_IUT">1</Param>
                                                <Param name="Interface_Servico">0</Param>
                                                            <Param name="M">1</Param>
                                                        <Param name="Id_con">1</Param>
                                                <Param name="Comprimento">1</Param>
                                                <Param name="Informacao_teste">1</Param>
                                            </Event>
                                           </Input>
                                    <!-- Exp: resultado esperado -->
                                     <Exp type=" "...> </Exp type>
                                <!-- Resultado a ser preenchido na execução dos testes -->
                                        <Result type=" "...> </Result type>
                            <!-- Verdict: tag opcional, a ser preenchida na execução dos testes -->
                                        <Verdict>  </Verdict>
                                            </TestCase>


                <TestCase name="testcase_injection_faul" objective="duplicidade">
                                        <Input>
                                    <Event name="init">
```


80

```
                    <Param name="get_host_by_name">hostname</Param>
                    <Param name="get_protocolo_by_name">tcp</Param>
                    <Param name="get_protocolo_by_name">rs422</Param
                    <Param name="get_service_by_name">dev</Param>
                    <Param name="get_port_AF">1971</Param>
                    <Param name="get_port_Experimento">1973</Param>
                </Event>
            <Event name="injection_1">
                                <!--Mensagem de Controle -->
                            <Param name="TestCase">1</Param>
                        <Param name="FluxoControl">1</Param>
                                <Param name="Modo">1</Param>
                        <Param name="Id_Conexao">1</Param>
                        <Param name="Campo_extra"> </Param>
                                        <!--Mensagem de Dados -->
                                    <Param name="Id">1</Param>
                                <Param name="Id_IUT">1</Param>
                    <Param name="Interface_Servico">0</Param>
                                    <Param name="M">1</Param>
                                <Param name="Id_con">1</Param>
                            <Param name="Comprimento">1</Param>
                    <Param name="Informacao_teste">1</Param>

                                    <Param name="Id2">1</Param>
                                <Param name="Id_IUT2">1</Param>
                    <Param name="Interface_Servico2">0</Param>
                                    <Param name="M2">1</Param>
                                <Param name="Id_con2">1</Param>
                            <Param name="Comprimento2">1</Param>
                    <Param name="Informacao_teste2">1</Param>
                </Event>
            </Input>
        <!-- Exp: resultado esperado -->
            <Exp type=" "...> </Exp type>
    <!-- Resultado a ser preenchido na execução dos testes -->
            <Result type=" "...> </Result type>
<!-- Verdict: tag opcional, a ser preenchida na execução dos testes -->
            <Verdict>  </Verdict>
        </Testcase>

    <TestCase name="testcase_injection_faul2" objective="omissao">
            <Input>
        <Event name="init">
                    <Param name="get_host_by_name">hostname</Param>
                    <Param name="get_protocolo_by_name">tcp</Param>
                    <Param name="get_protocolo_by_name">rs422</Param
                    <Param name="get_service_by_name">dev</Param>
                    <Param name="get_port_AF">1971</Param>
```

```
                    <Param name="get_port_Experimento">1973</Param>
                </Event>
            <Event name="Pacote_1">
                        <!--Mensagem de Controle -->
                    <Param name="TestCase">1</Param>
                <Param name="FluxoControl">1</Param>
                    <Param name="Modo">1</Param>
                <Param name="Id_Conexao">1</Param>
                <Param name="Campo_extra"> </Param>
                        <!--Mensagem de Dados -->
                    <Param name="Id"> </Param>
                <Param name="Id_IUT"> </Param>
            <Param name="Interface_Servico"> </Param>
                    <Param name="M"> </Param>
                <Param name="Id_con"> </Param>
            <Param name="Comprimento"> </Param>
            <Param name="Informacao_teste"> </Param>
        </Event>
    </Input>
<!-- Exp: resultado esperado -->
<Exp type=" "...> </Exp type>
<!-- Resultado a ser preenchido na execução dos testes -->
<Result type=" "...> </Result type>
<!-- Verdict: tag opcional, a ser preenchida na execução dos testes -->
<Verdict>  </Verdict>
    </TestCase>
</TestSuite>
```


82